Course #2:

Deep Learning, from MLP to CNN

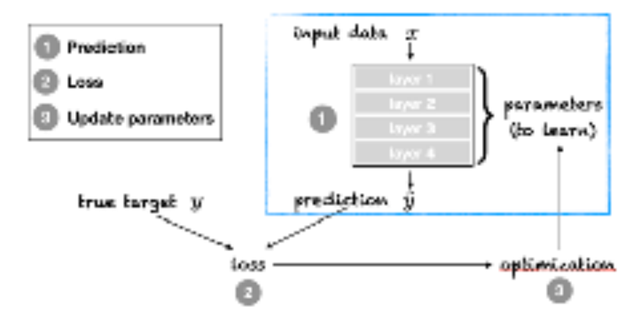
Roadmap

- Recap from course #1
- MLP and Image classification as a case study
- CNN: basic principles
- Application to image classification
- Classic CNN architectures
- Auco-encoders

Recap from Course #1
Things to know

- Supervised vs. unsupervised learning
- Training and test dataset
- Training loss
- Model

Guidelines to implement Deep Learning schemes



1. Problem formulation (inputs/outputs)
2. Data collection (cf. supervised vs. non-supervised)
3. Definition of performance metrics
4. Selection of neural architectures (at least 2 models)
5. Selection of a training loss
6. Split dataset into training / validation / test datasets
7. Train the selected models from the training dataset and save the best models onto the validation dataset
8. Benchmark the performance of the trained models onto the test dataset
9. Update/iterate 4-5-6-7-8

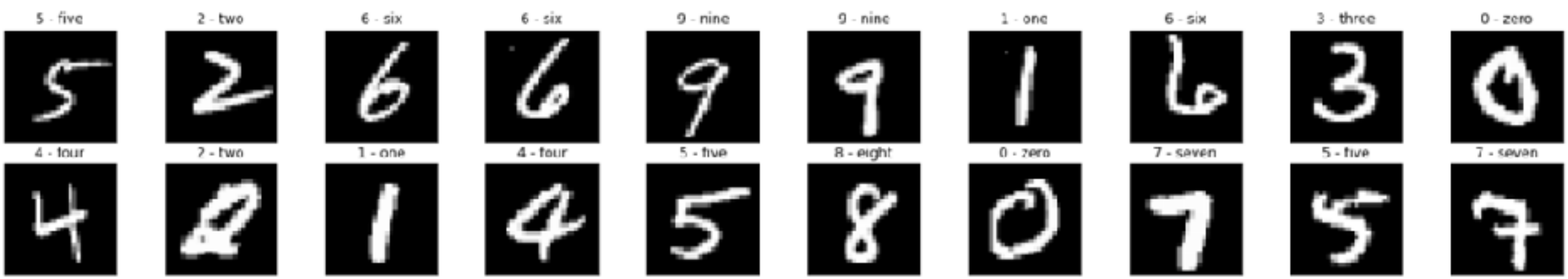
Image classification case-study

Let's go

https://github.com/CIA-Oceanix/DLCourse_MOi_2022/blob/main/notebooks/notebook_MNIST_classification_MLP_with_correction.ipynb

Image classification case-study with pytorch

1. Problem formulation (inputs/outputs)



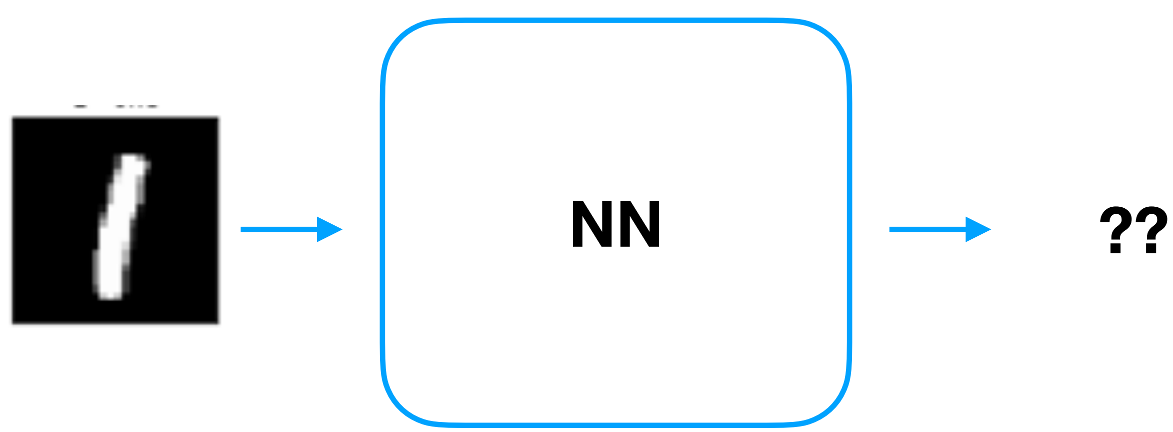


Image classification case-study with pytorch

2. Data collection

```
train_data = datasets.MNIST(root = 'data', train = True, download = True, transform = transform)
test_data = datasets.MNIST(root = 'data', train = False, download = True, transform = transform)
```

3. Performance metrics

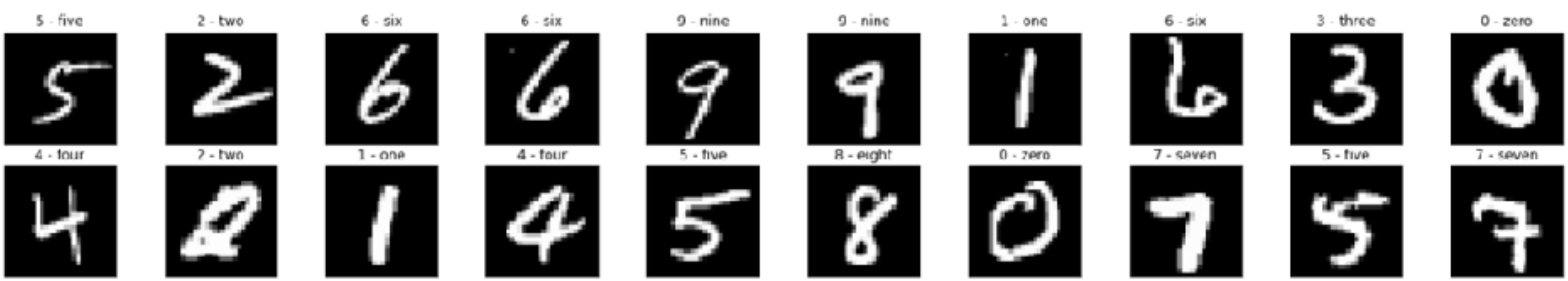


Image classification case-study with pytorch

4. Neural architecture

```
import torch.nn as nn
import torch.nn.functional as F

class MLP(nn.Module):
    def __init__(self): # FUNCTION TO BE COMPLETED
        super(MLP,self).__init__()
        hidden_1, hidden_2 = 512, 256
        self.fc1 = nn.Linear(28*28, hidden_1)
        self.fc2 = nn.Linear(hidden_1,hidden_2)
        self.fc3 = nn.Linear(hidden_2,10)
        self.dropout = nn.Dropout(0.2)

    def forward(self,x): # FUNCTION TO BE COMPLETED
        x = x.view(-1,28*28)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)
        x = F.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
        return x
```

5. Training loss

```
criterion = nn.CrossEntropyLoss() # TO DO
```

Model complexity ?

Image classification case-study with pytorch

6. Split dataset into training / validation / test datasets

```
batch_size = 20
valid_size = 0.2
train_size = 0.3

def create_data_loaders(batch_size, valid_size, train_data, test_data): # FUNCTION TO BE COMPLETED

  num_train = len(train_data)
  indices = list(range(num_train))
  np.random.shuffle(indices)
  nb_train = int( np.floor(train_size * num_train ))
  split = int(np.floor(valid_size * num_train))
  train_index, valid_index = indices[split:nb_train], indices[:split]

  train_sampler = SubsetRandomSampler(train_index)
  valid_sampler = SubsetRandomSampler(valid_index)

  train_loader = torch.utils.data.DataLoader(train_data, batch_size = batch_size, sampler = train_sampler
  valid_loader = torch.utils.data.DataLoader(train_data, batch_size = batch_size, sampler = valid_sampler
  test_loader = torch.utils.data.DataLoader(test_data, batch_size = batch_size)

  return train_loader, valid_loader, test_loader
```

Image classification case-study with pytorch

7. Model training

```
optimizer = torch.optim.SGD(model_1.parameters(),lr = 0.01)
```

```
n_epochs = 30

def training(n_epochs, train_loader, valid_loader, model, criterion, optimizer):

  train_losses, valid_losses = [], []
  valid_loss_min = np.Inf

  for epoch in range(n_epochs):
      train_loss, valid_loss = 0, 0

      model.train()
      for data, label in train_loader:
          data = data.to(device=device, dtype=torch.float32)
          label = label.to(device=device, dtype=torch.long)
          optimizer.zero_grad()
          output = model(data)
          loss = criterion(output, label)
          loss.backward()
          optimizer.step()
          train_loss += loss.item() * data.size(0)
```

Image classification case-study

Go and run the notebook

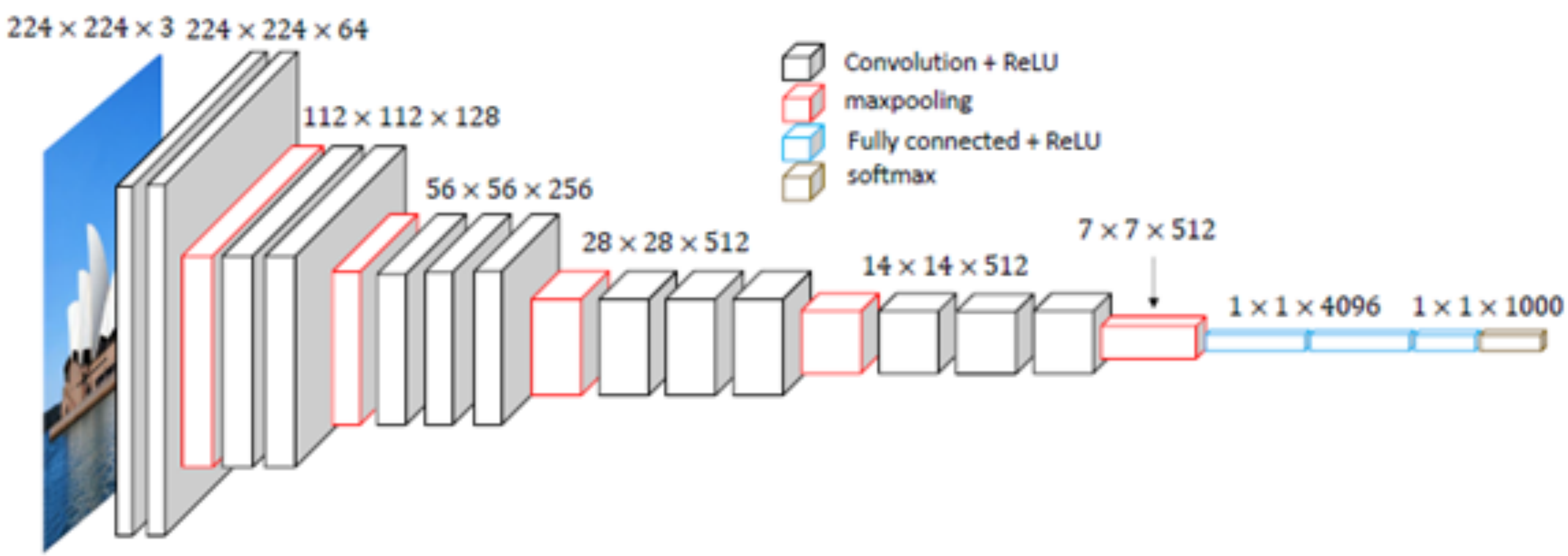
Questions :

What is the effect of the dropout layer in the MLP architecture ?

Convolutional Neural Networks

State-of-the-art NNs in computer vision

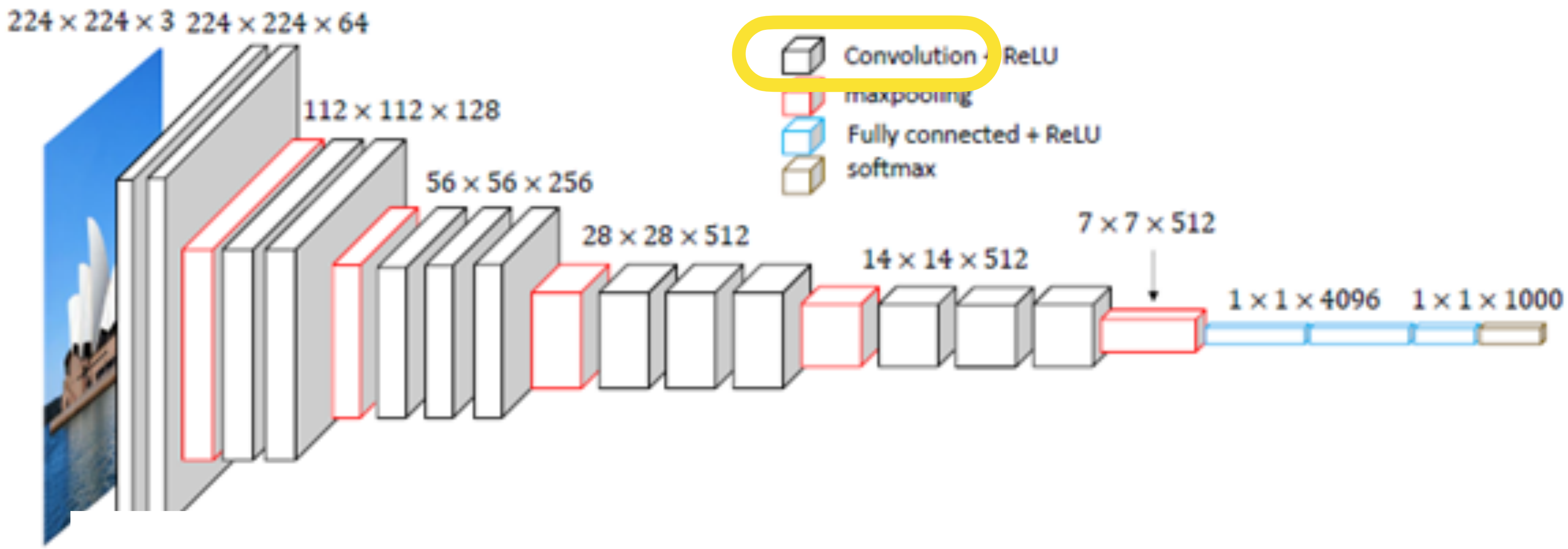
DL models are (in general) feedforward models. VGG16 as an illustration



Elementary components

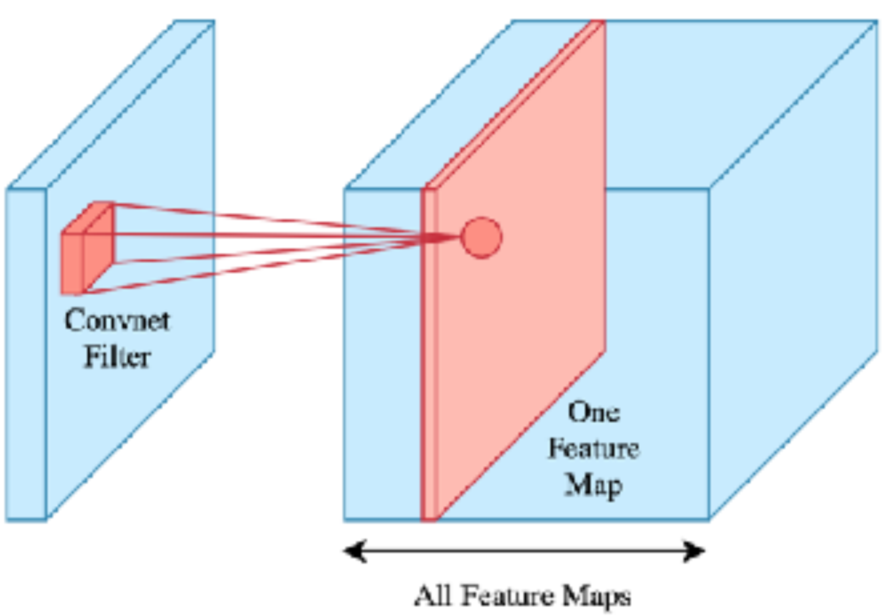
- Convolution layers
- Activation layers
- Pooling layers
- Dense layers

Basics of DL models



Elementary components

- Convolution layers
- Activation layers
- Pooling layers
- Dense layers



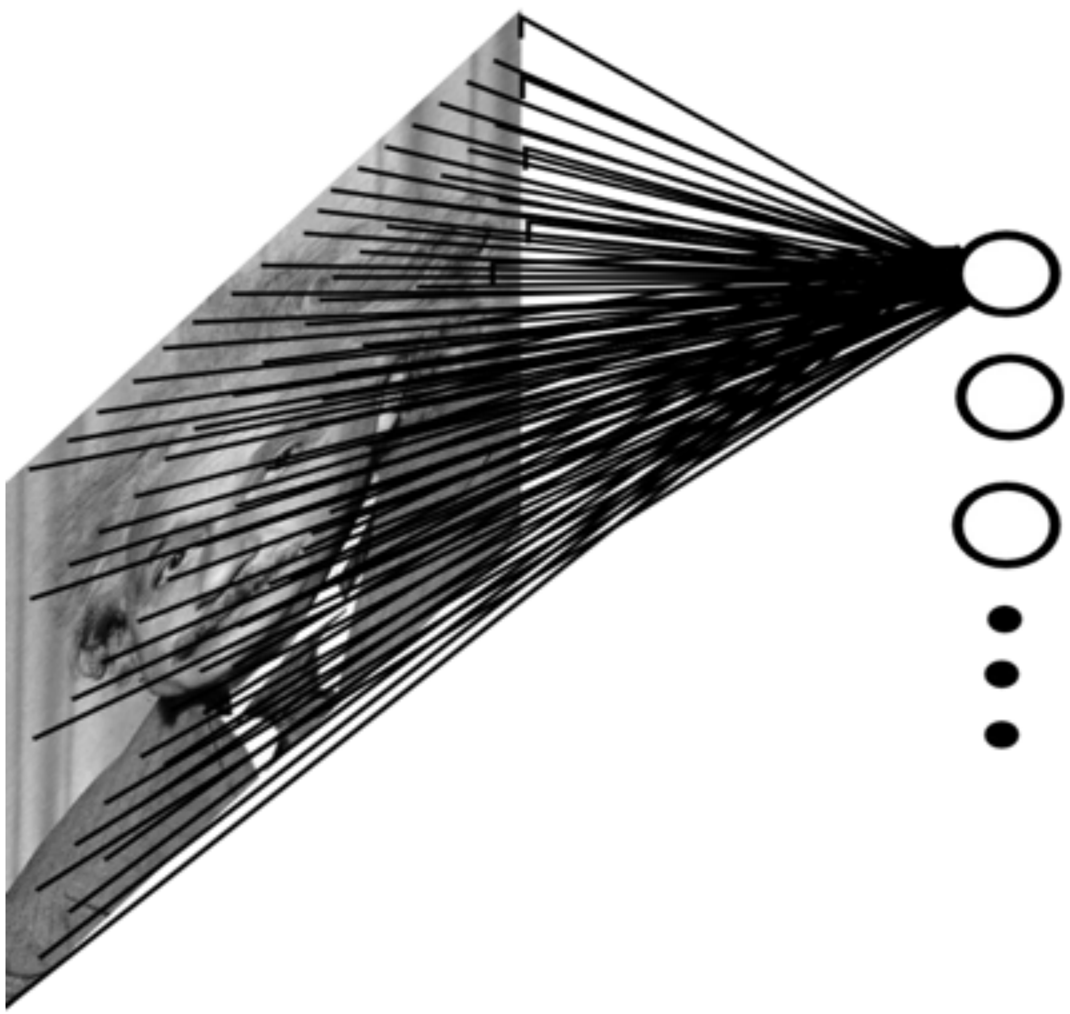

```
torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0,
dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)
```

https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

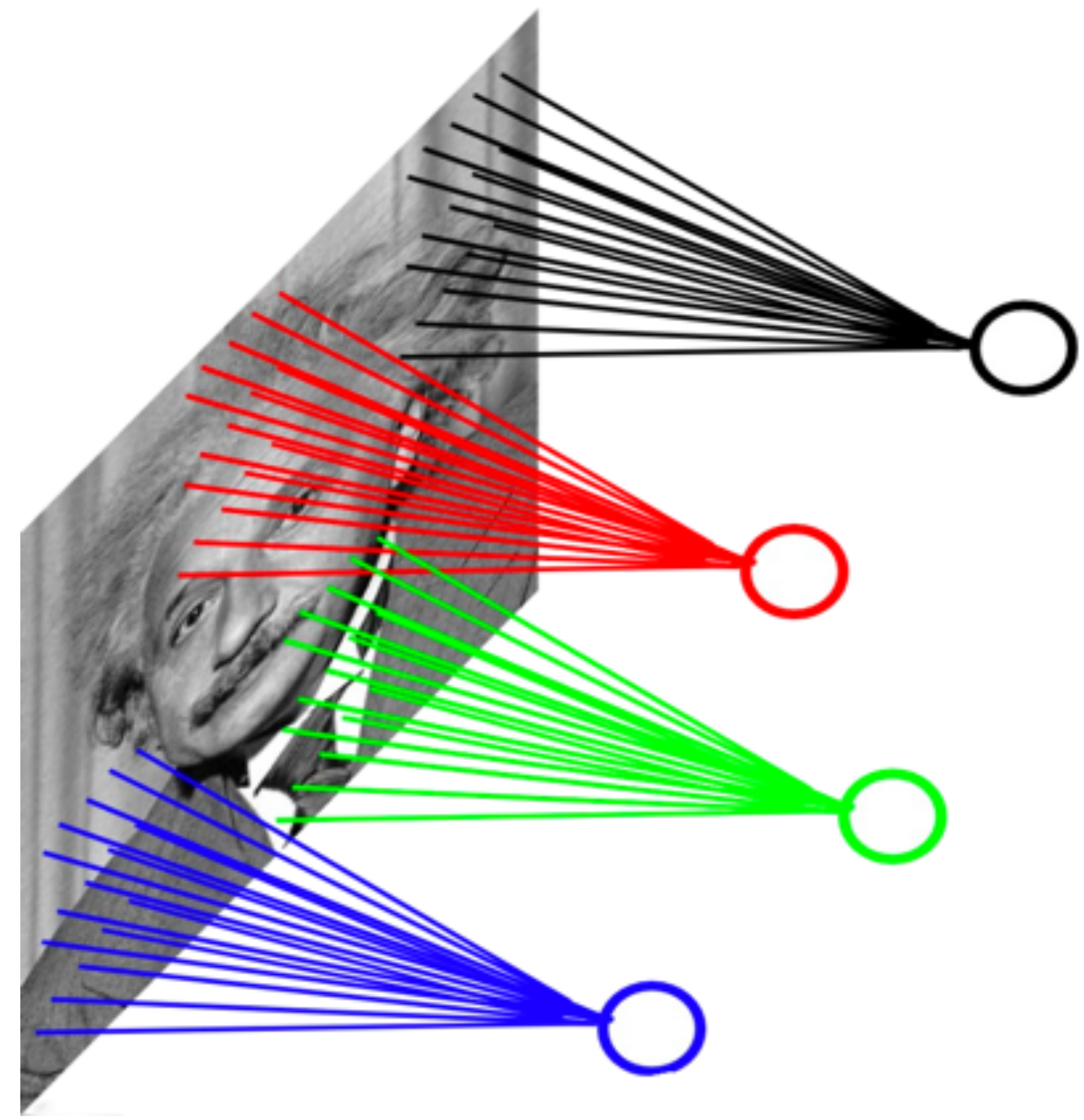
Number of parameters ?

Independent on the sizes of the input and output layer

Dense layer vs Conv layer

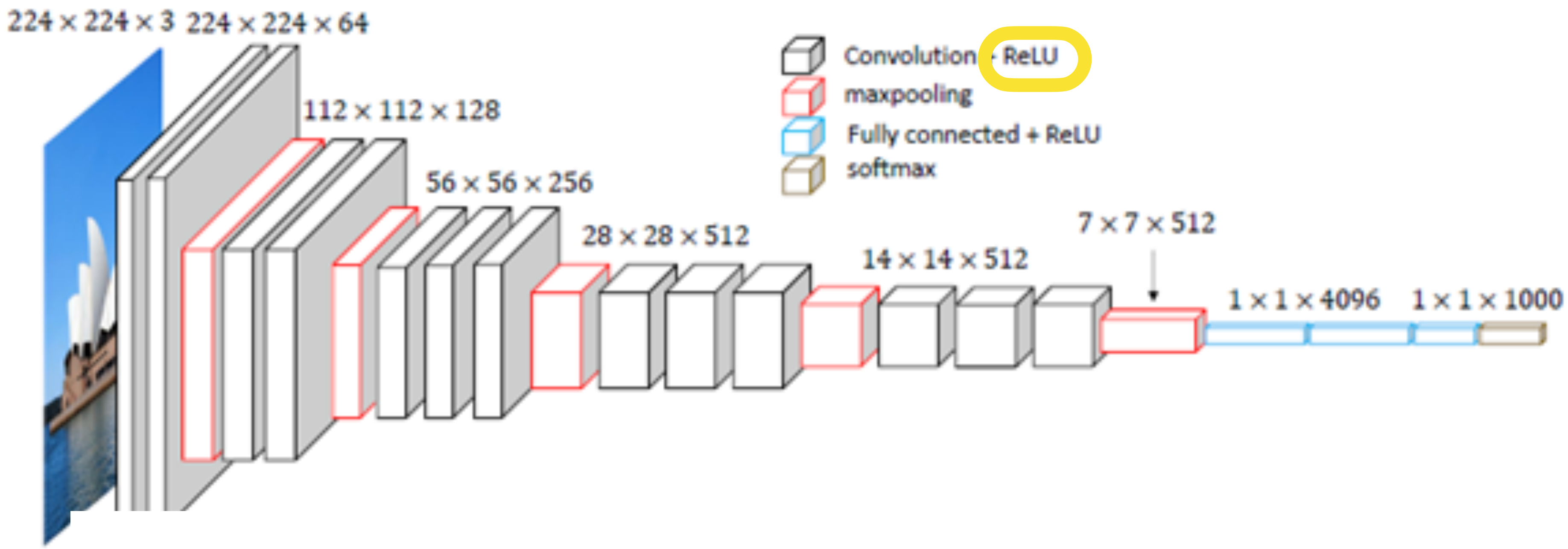


Dense layer



Convolutional layer

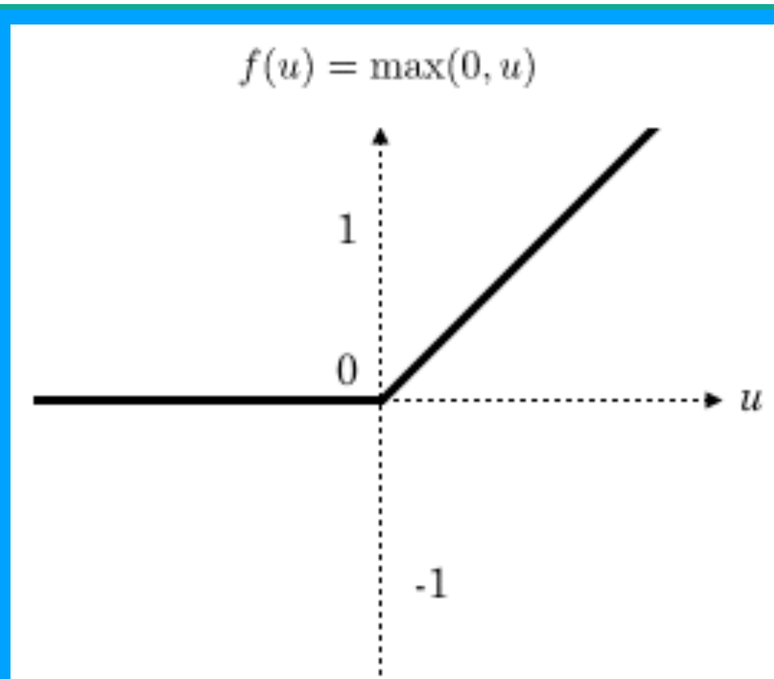
Basics of DL models



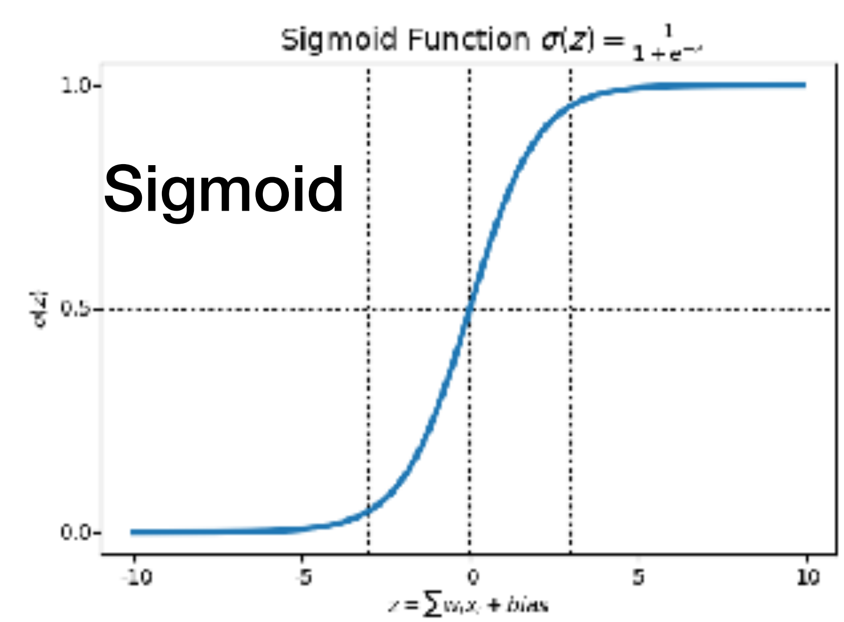
Elementary components

- Convolution layers
- Activation layers
- Pooling layers
- Dense layers

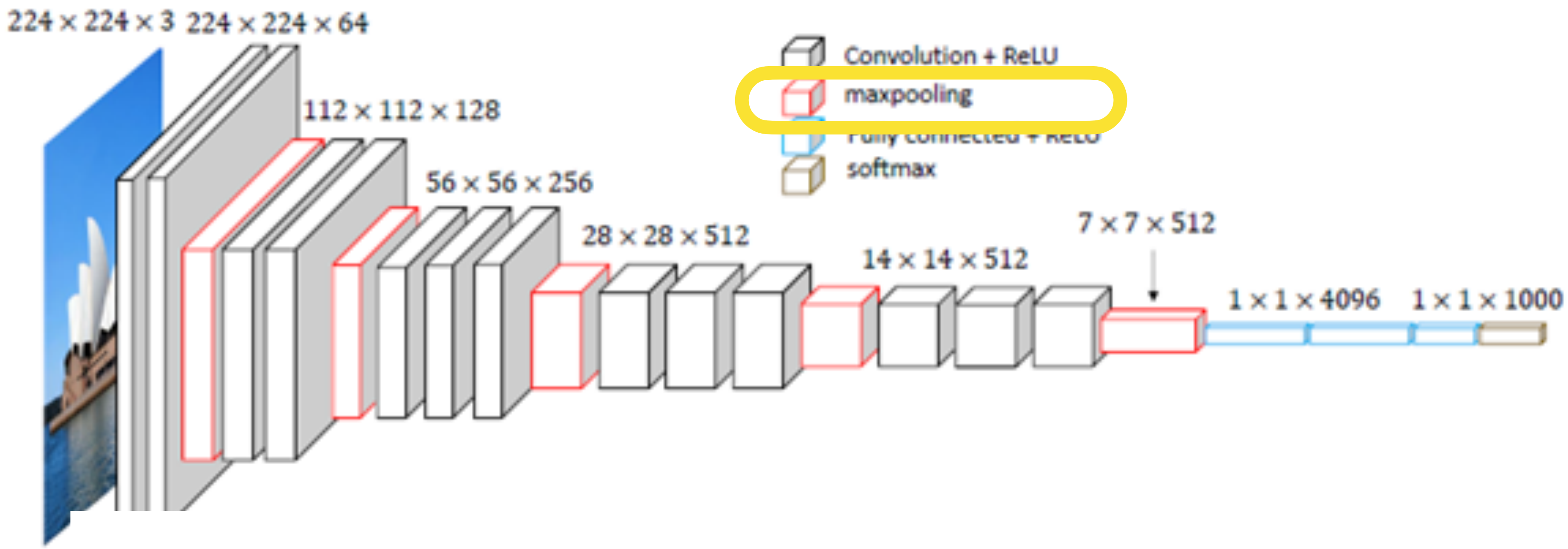
ReLU (Rectified Linear Unit)



Sigmoid

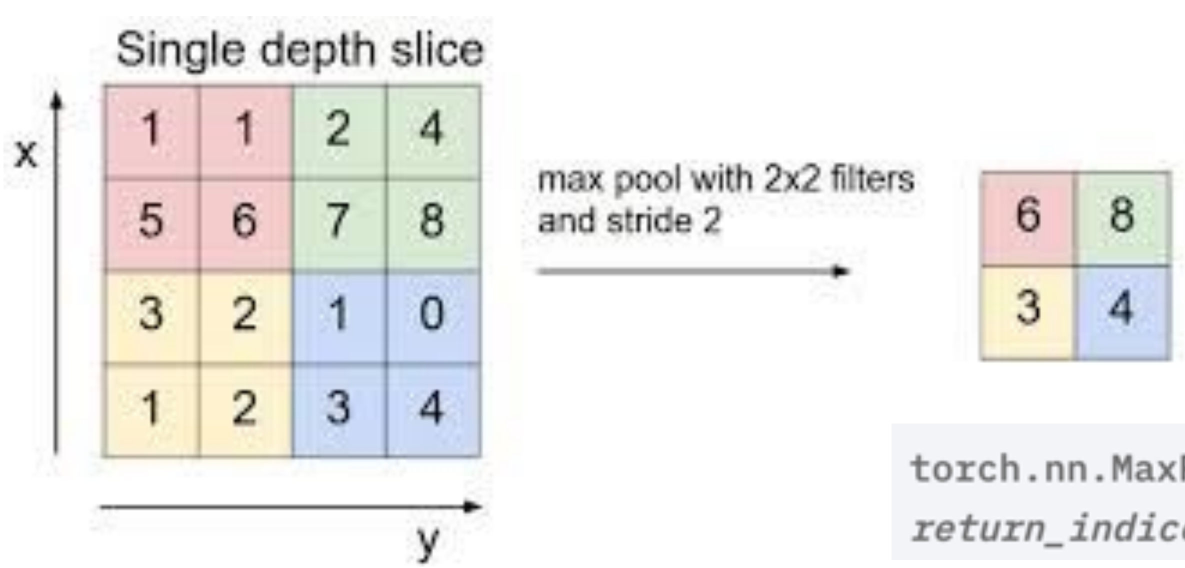


Basics of DL models

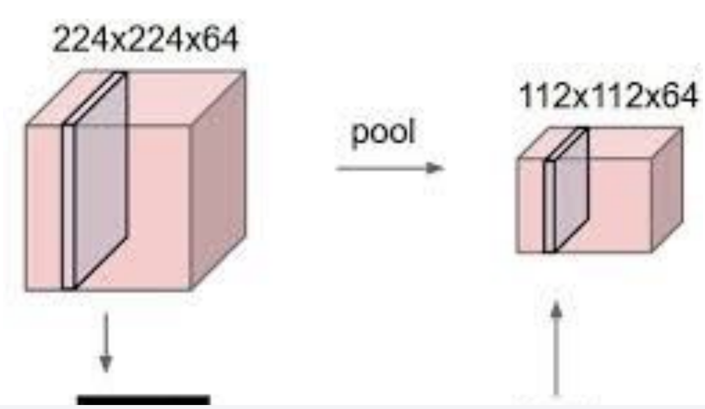


Elementary components: Convolution layers | Activation layers | Pooling layers | Dense layers

An example of max-pooling operator

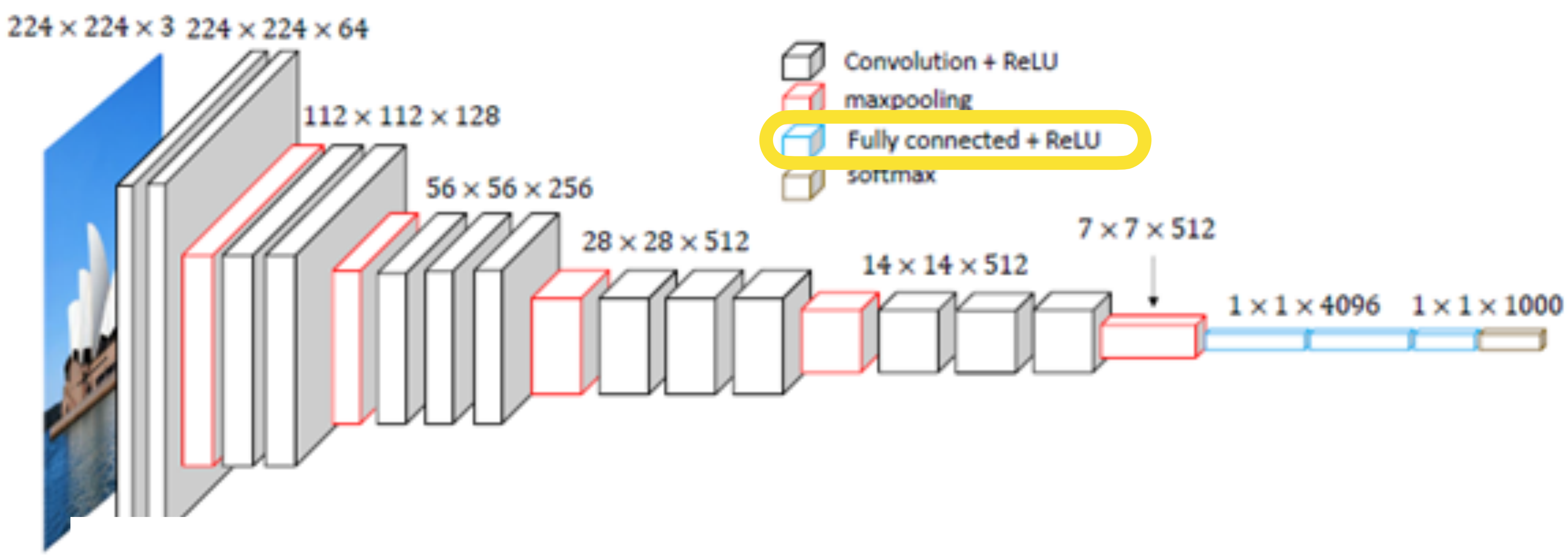


Pooling downsamples the input layer




```
torch.nn.MaxPool2d(kernel_size, stride=None, padding=0, dilation=1,
return_indices=False, ceil_mode=False) [SOURCE]
```

Basics of DL models



Elementary components

Convolution layers | Activation layers | Pooling layers | **Dense layers**

Dense layers
or
Fully-connected (FC) layer

as in a classic MLP

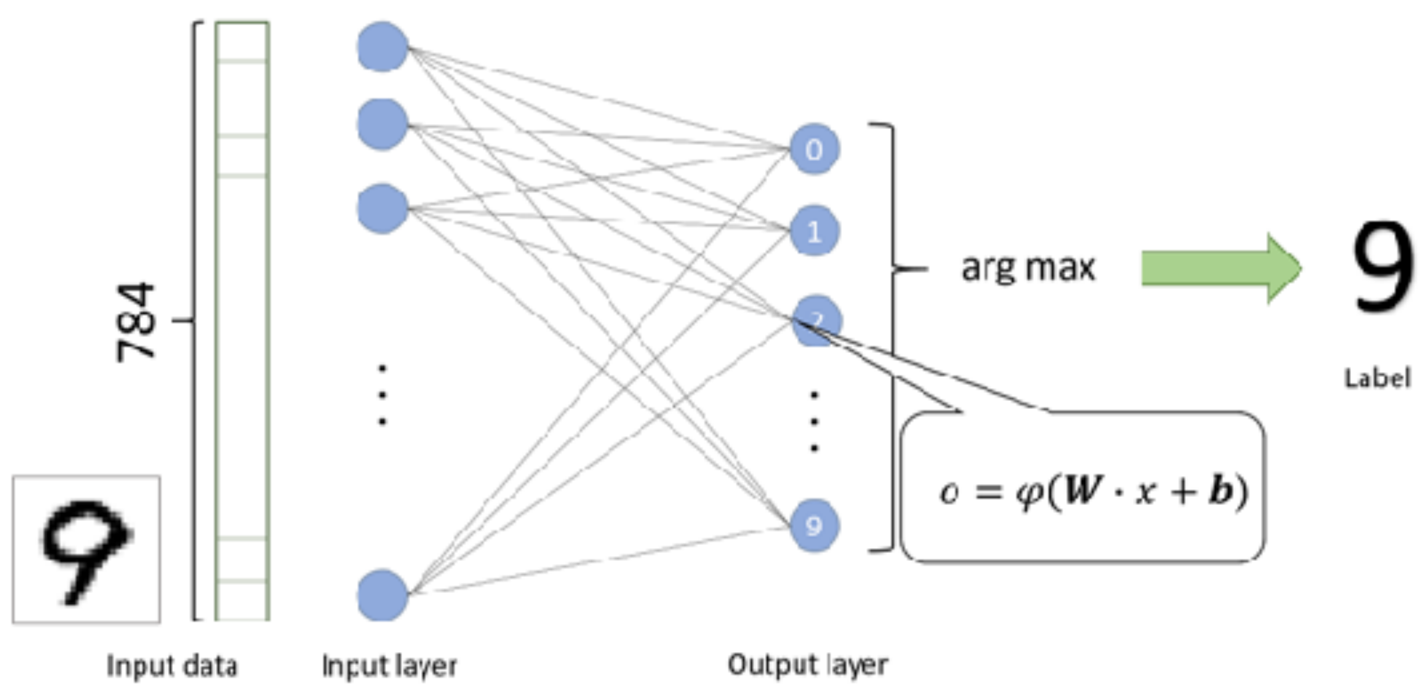
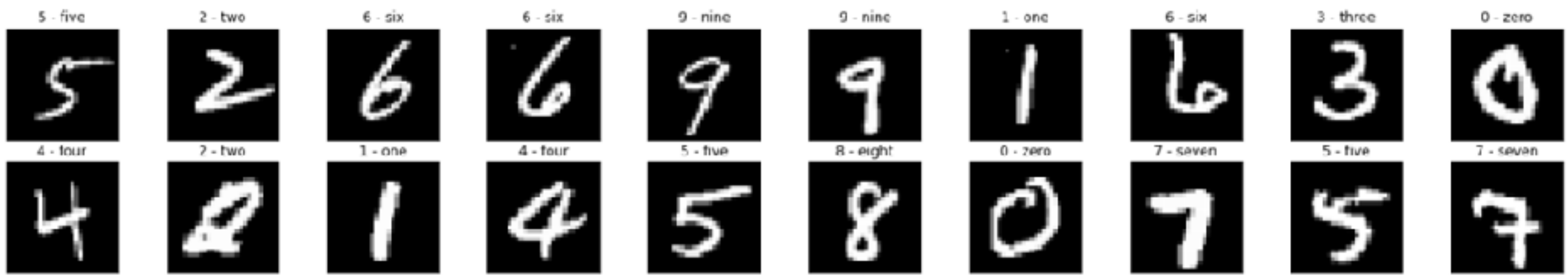
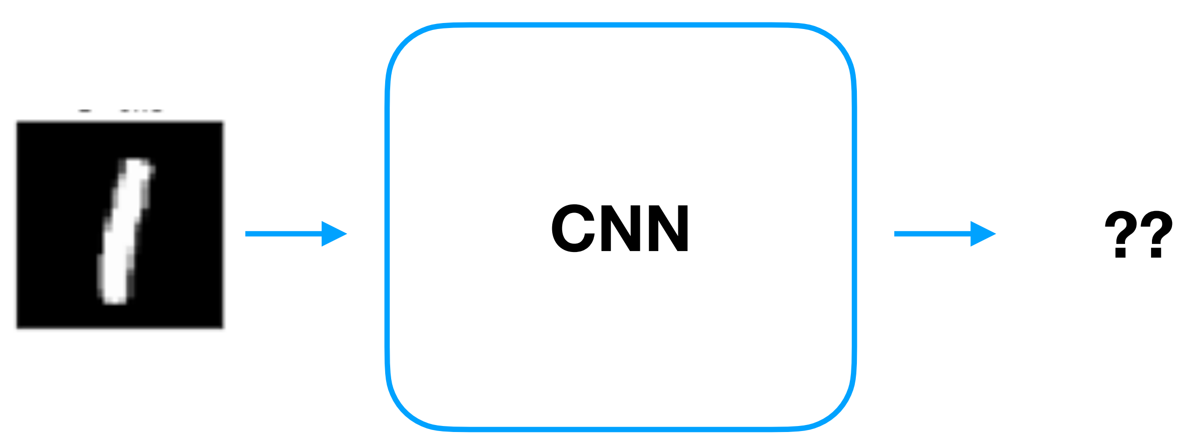


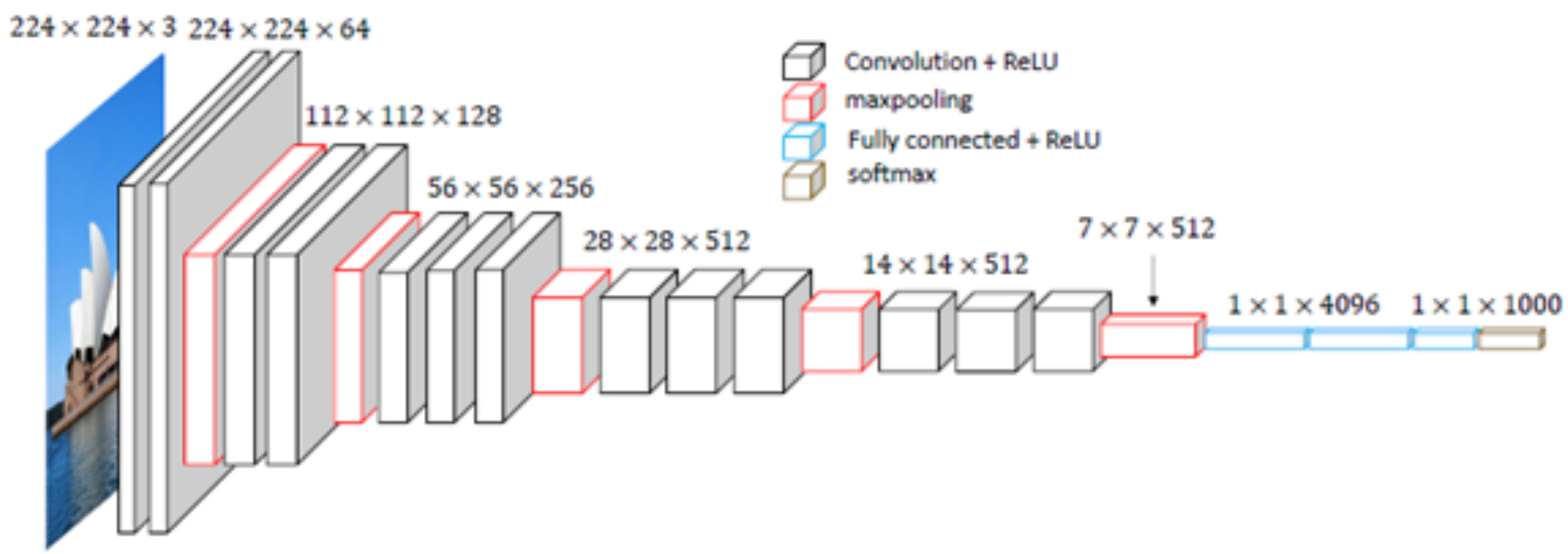
Image classification case-study with CNN





https://github.com/CIA-Oceanix/DLCourse_MOi_2022/blob/main/notebooks/notebook_MNIST_classification_MLP_CNN_students.ipynb

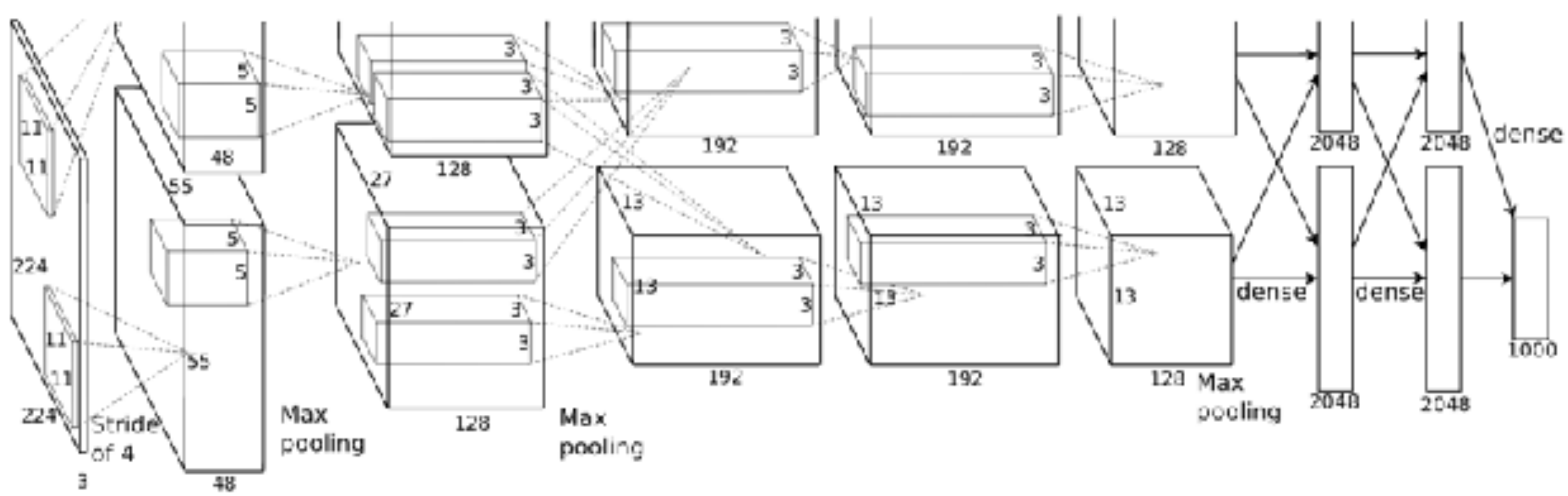
Examples of DL models for object recognition (2010-2020)



VGG16
(<100M of parameters)

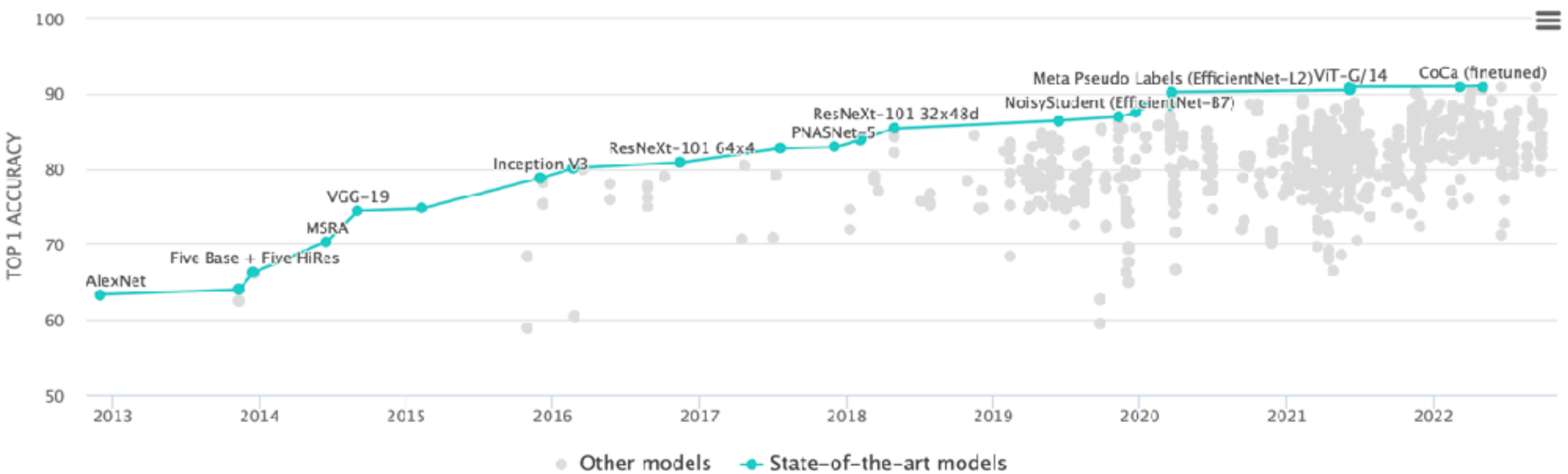
Google `
inception
(5M of parameters)

Convolution
Pooling
Softmax
Other



AlexNet
(60M of parameters)

DL and Benchmarking
(Data Challenges)

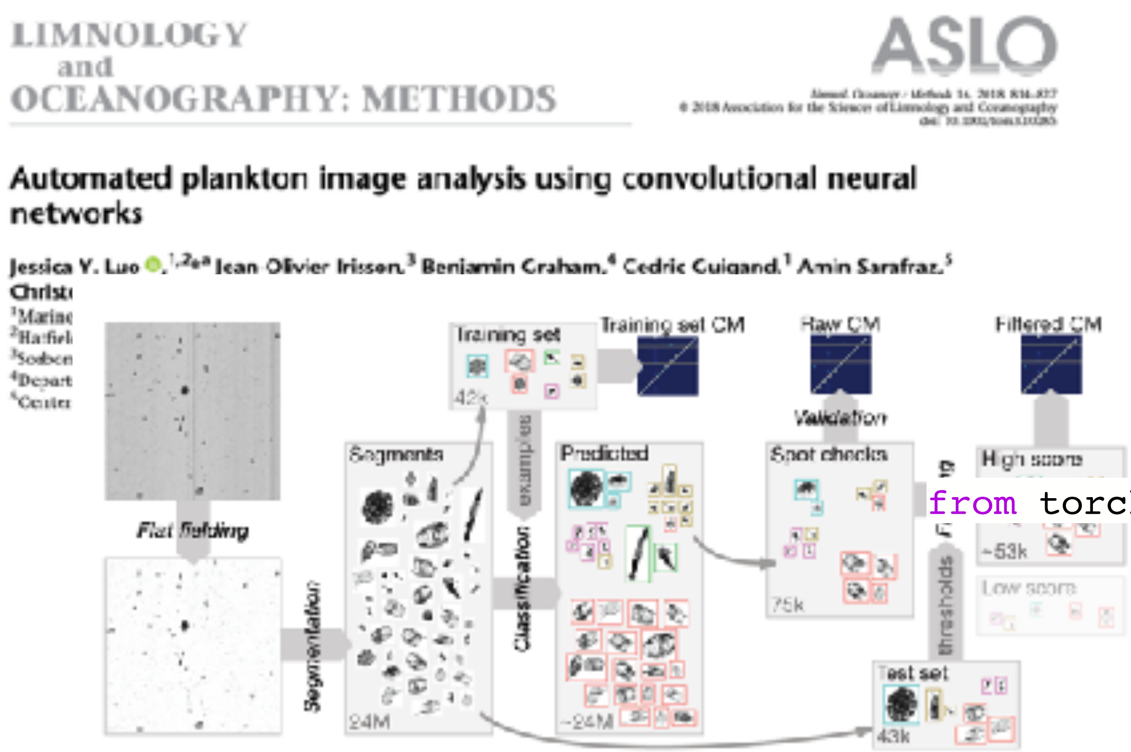


https://paperswithcode.com/sota/image-classification-on-imagenet

# of object classes: 1000
# of images > 1.2 M

Best accuracy score: ~91%
State-of-the-art architectures: CNN, Vision Transformers

CNN-based classification and Ocean Data

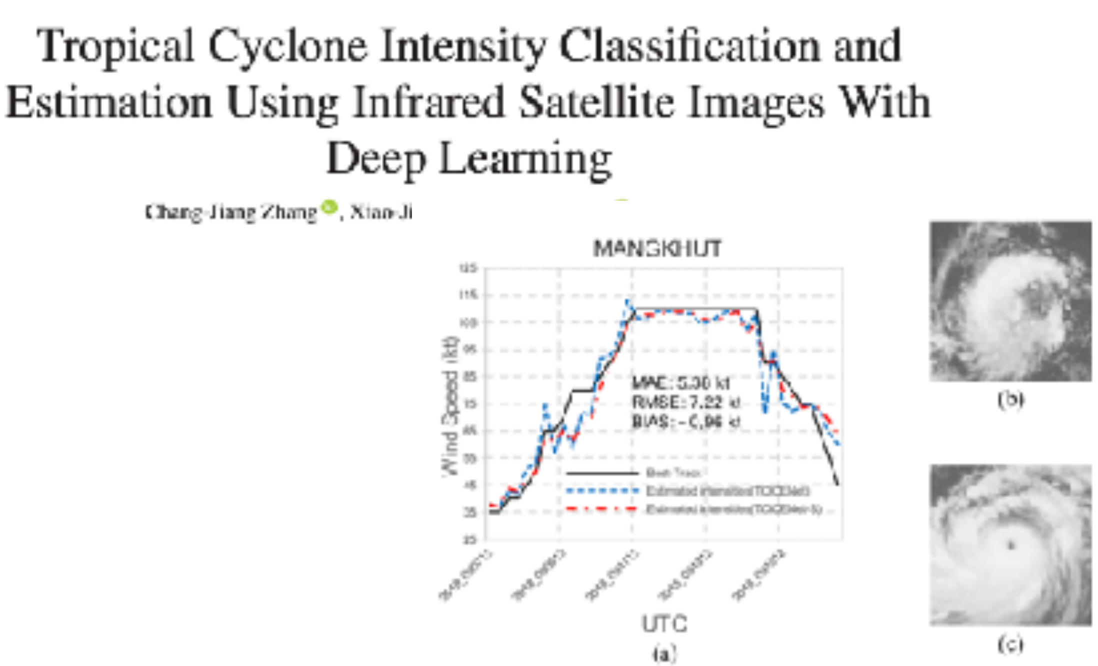

LIMNOLOGY and OCEANOGRAPHY: METHODS

ASLO

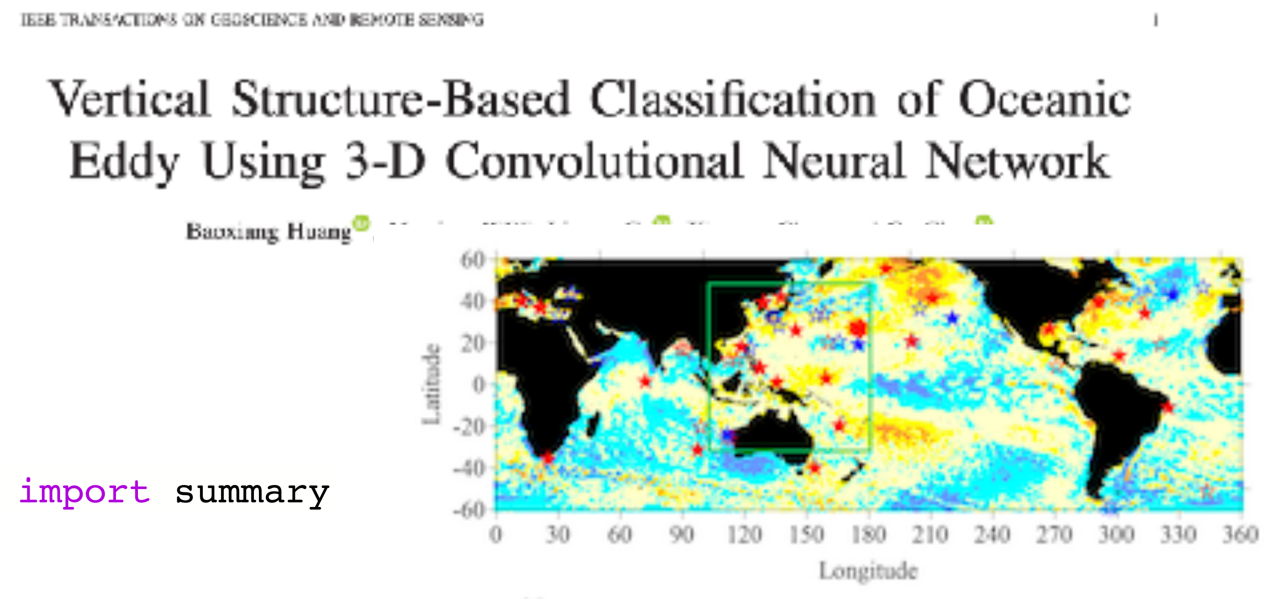
Automated plankton image analysis using convolutional neural networks

Jessica Y. Luo, Jean Olivier Irisson, Benjamin Graham, Cedric Guigand, Amin Sarafraz


```
from torchsummary import summary
```

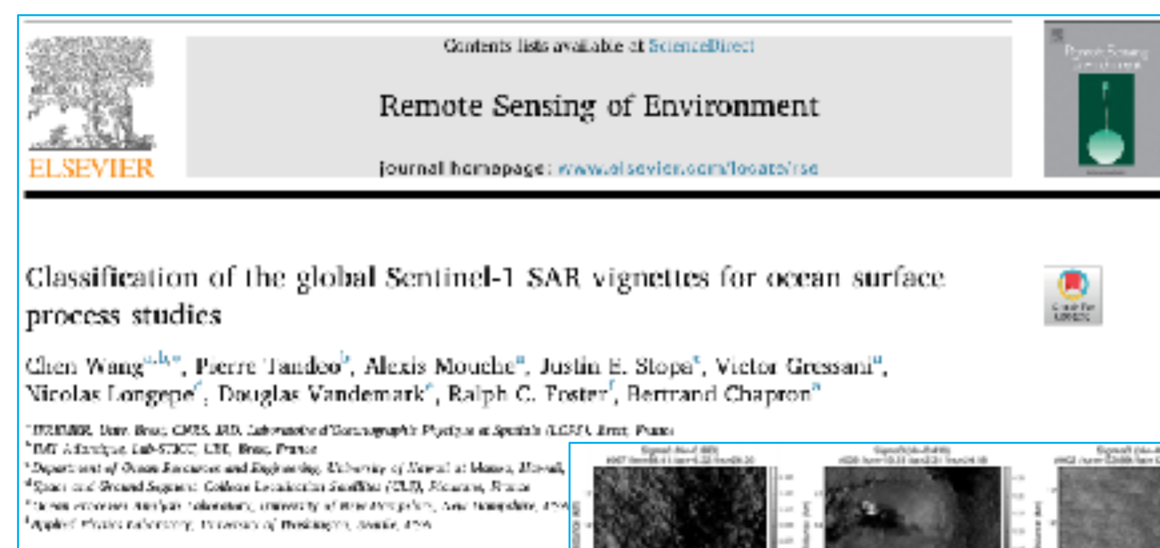

Tropical Cyclone Intensity Classification and Estimation Using Infrared Satellite Images With Deep Learning

Chang-Jiang Zhang, Xiao-Ji


IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING

Vertical Structure-Based Classification of Oceanic Eddy Using 3-D Convolutional Neural Network

Baoxiang Huang


Contents lists available at ScienceDirect

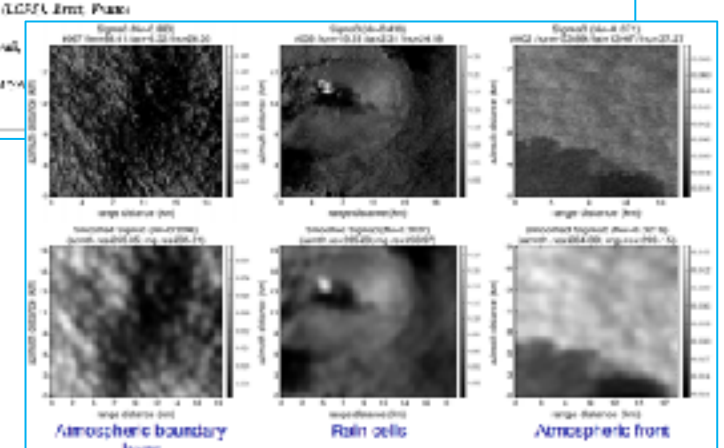
Remote Sensing of Environment

journal homepage: www.elsevier.com/locate/rse

ELSEVIER

Classification of the global Sentinel-1 SAR vignettes for ocean surface process studies

Chen Wang, Pierre Tandeo, Alexis Mouche, Justin E. Stopa, Victor Gressani, Nicolas Longepe, Douglas Vandemark, Ralph C. Foster, Bertrand Chapron



Fine-tuning from pre-trained models

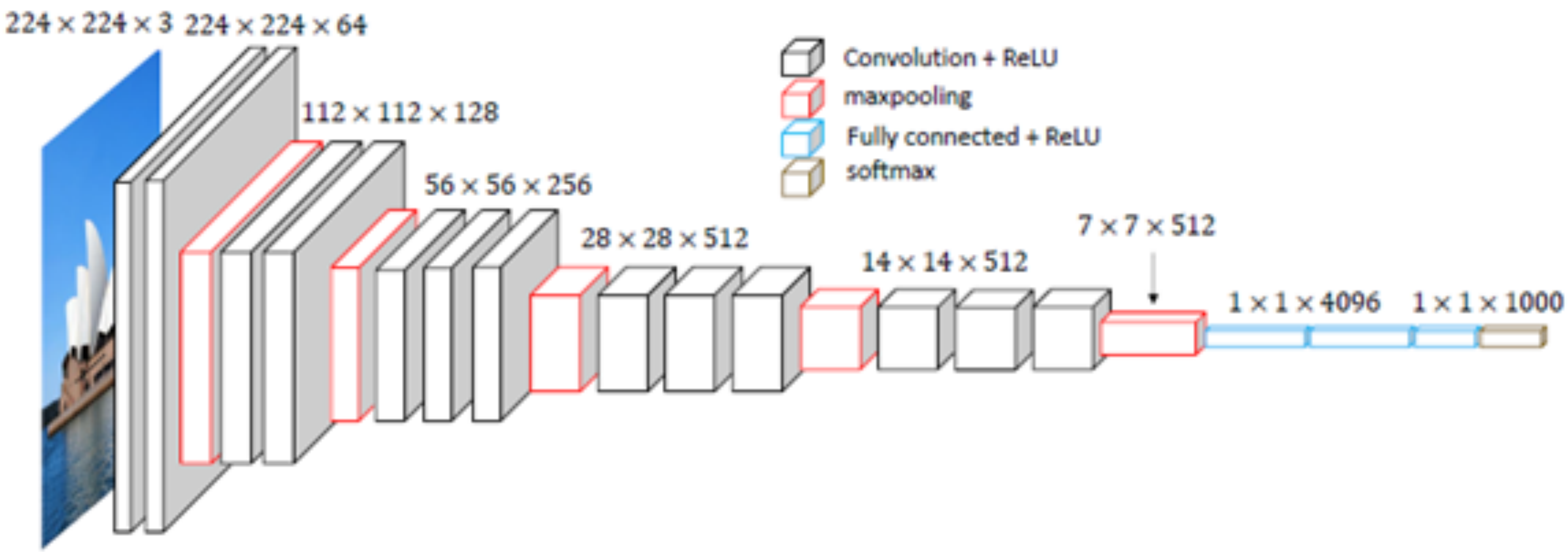


Figure 2: The architecture of VGG16 model .

General idea: the first layers involve generic feature extraction step and the last block can be regarded as a dataset-specific classification block.

Fine-tuning from pre-trained models

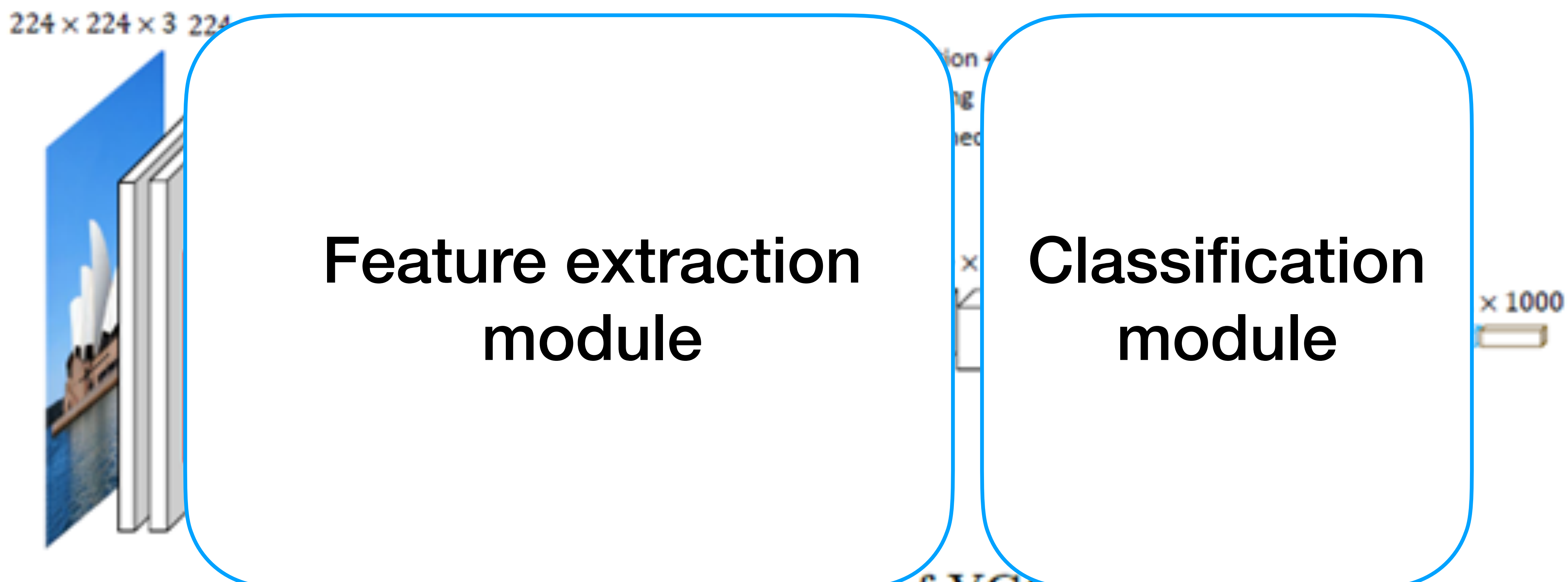
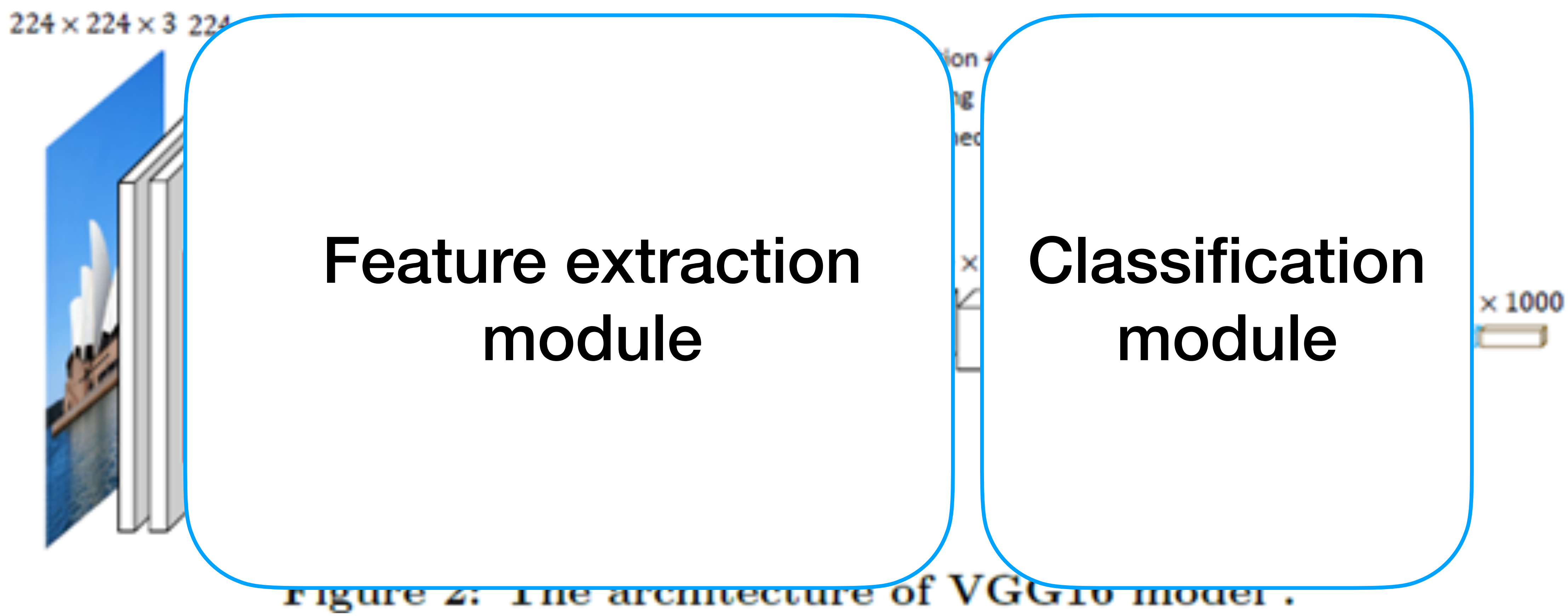


Figure 2: The architecture of VGG16 model .

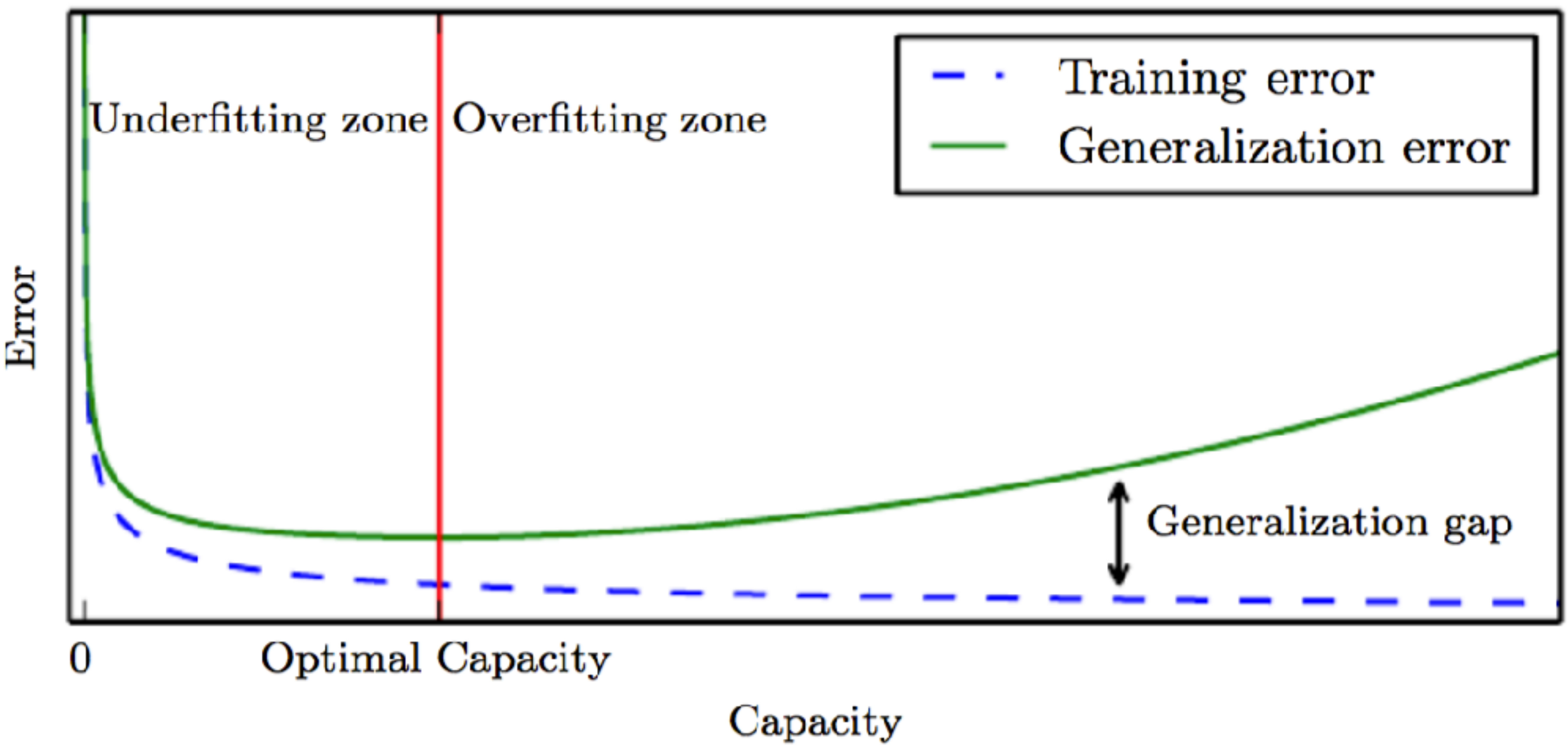
General idea: the first layers involve generic feature extraction step and the last block can be regarded as a dataset-specific classification block.

Fine-tuning from pre-trained models

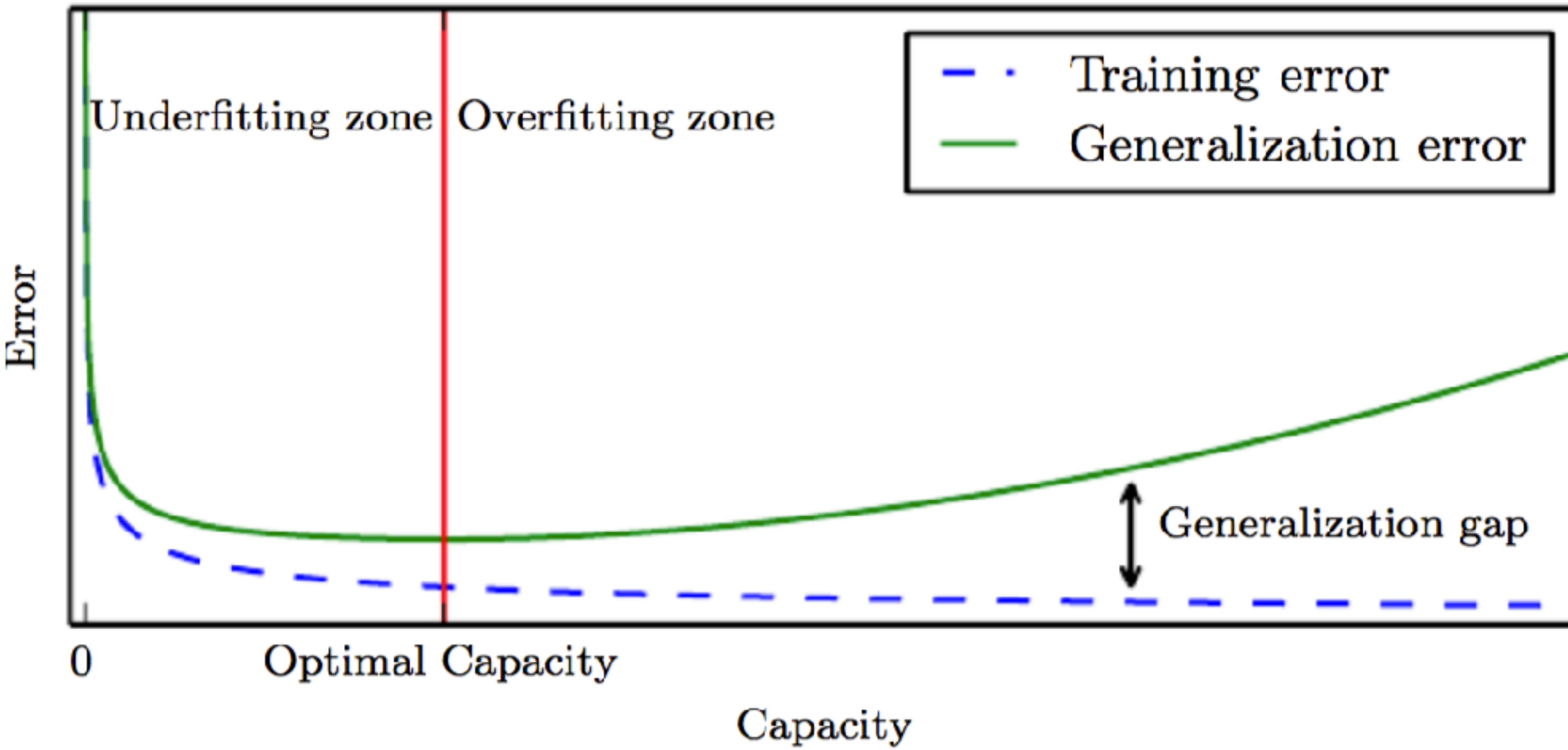


https://github.com/CIA-Oceanix/DLCourse_MOi_2022/blob/main/notebooks/notebook_MNIST_classification_MLP_CNN_TransferLearning_students.ipynb

Over-fitting



Over-fitting



Regularzation tricks to avoid overfitting

- Penalty terms In the training loss
- Data augmentation
- Dropout layers

Parameter norm penalization

- Regularized objective function:

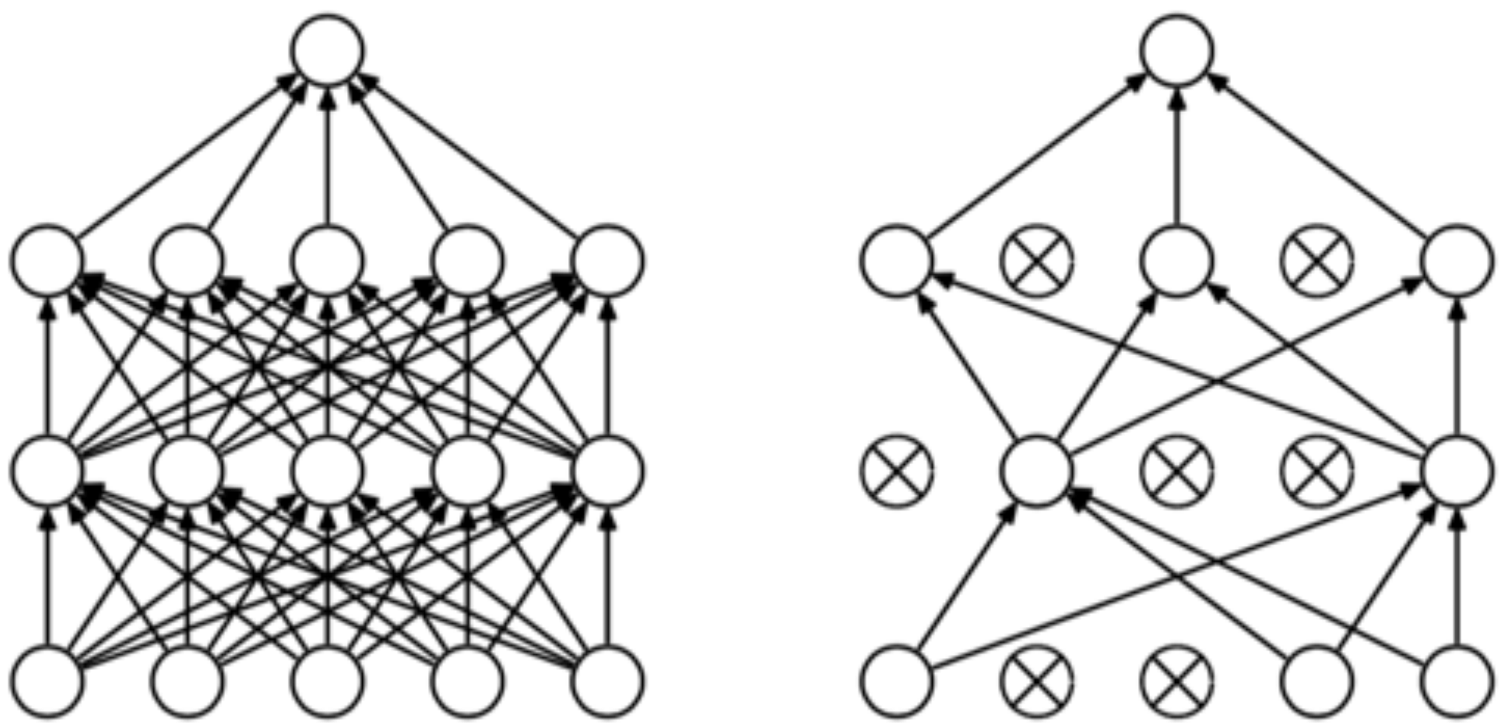
$$\tilde{J}(\theta) = J(\theta) + \alpha\Omega(\theta)$$

- L^2 norm: $\Omega(\theta) = \frac{1}{2}||w||_2^2$
- L^1 norm: $\Omega(\theta) = ||w||_1 = \sum_i |w_i|$

Data augmentation

- Purpose: improving model generalization error by training on more data
- Very efficient for object recognition
- How to:
  - apply (geometric) transformations on input data (such as translation, rotation, scaling for images).
  - noise injection

Dropout

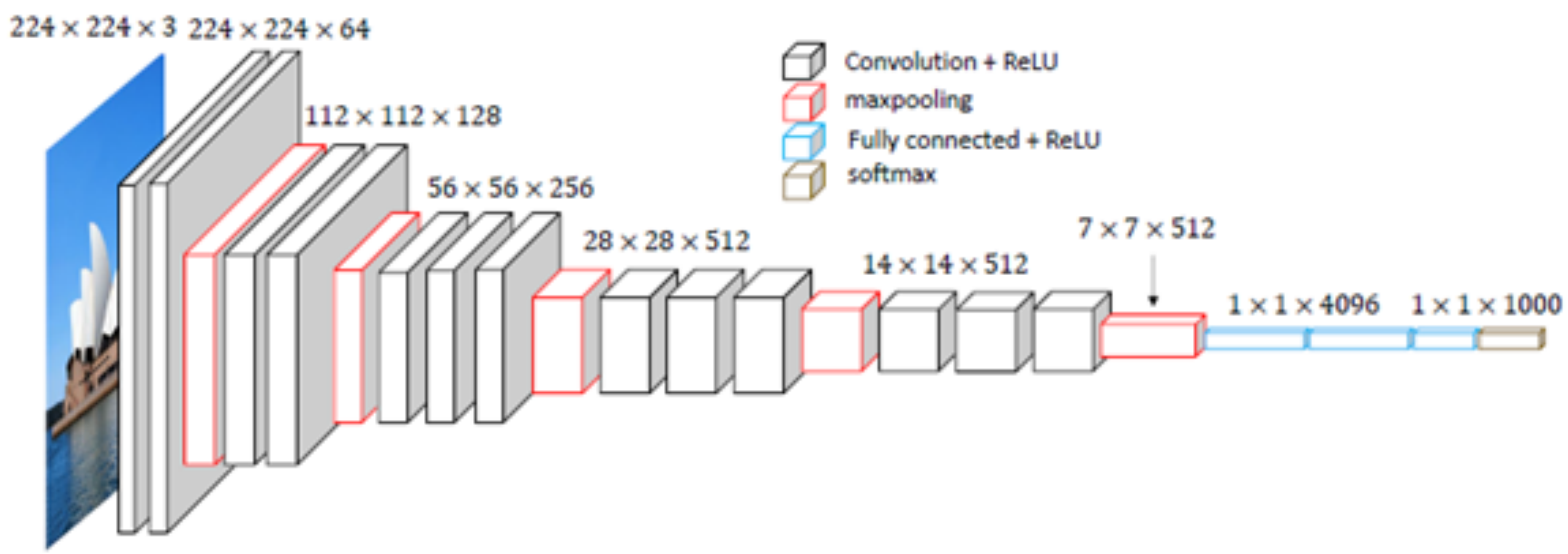


(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

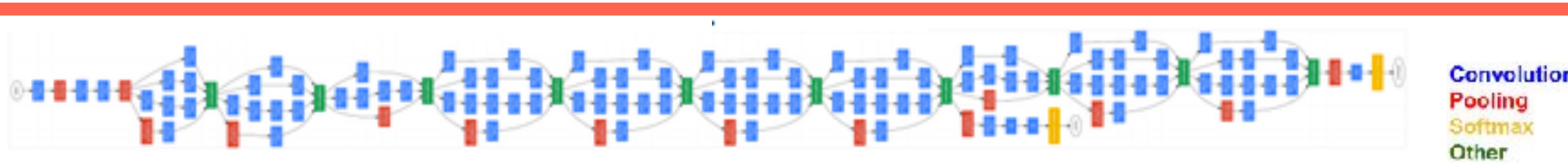
Srivastava et al., 2014

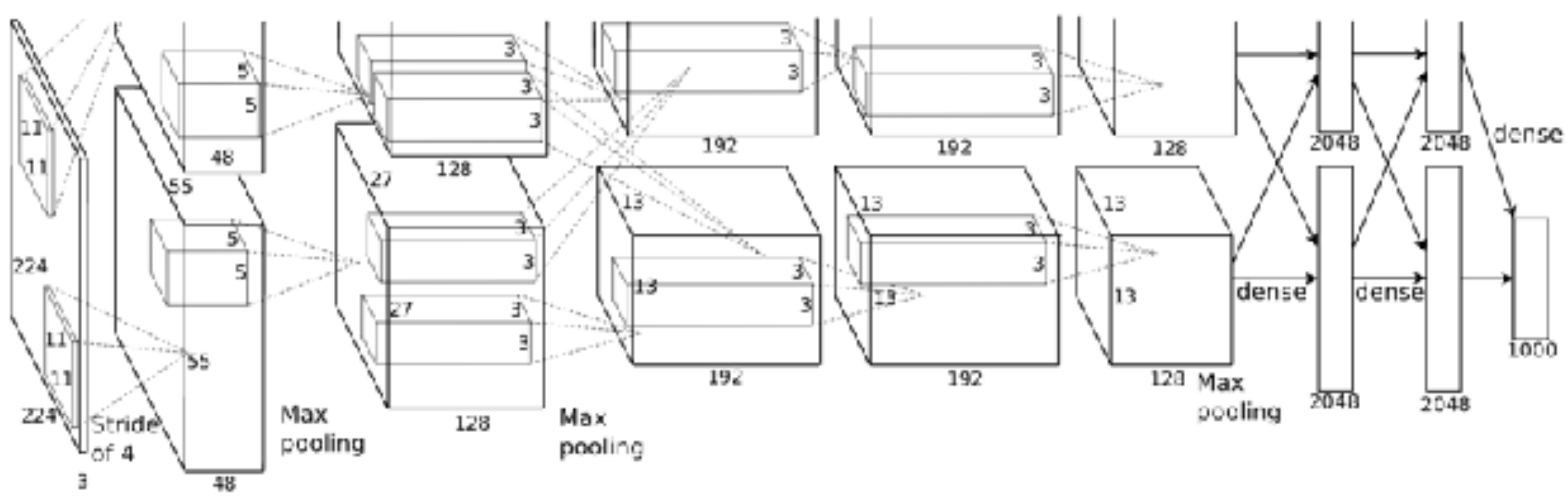
Examples of DL models for object recognition (2010-2020)



VGG16
(<100M of parameters)

Google `
inception
(5M of parameters)





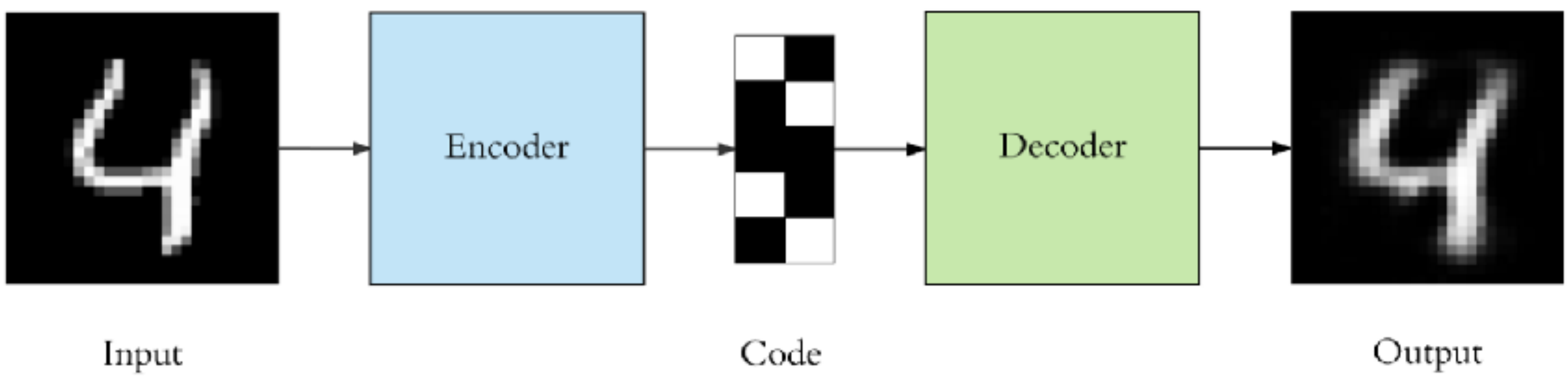
AlexNet
(60M of parameters)

Lecture. #2
Things to know (CNN)

- Convolution layers
- Pooling layers
- Activation layers
- Dropout layers
- Padding and stride
- Fine-tuning
- Over-fitting
- Data augmentation

Auto-encoders

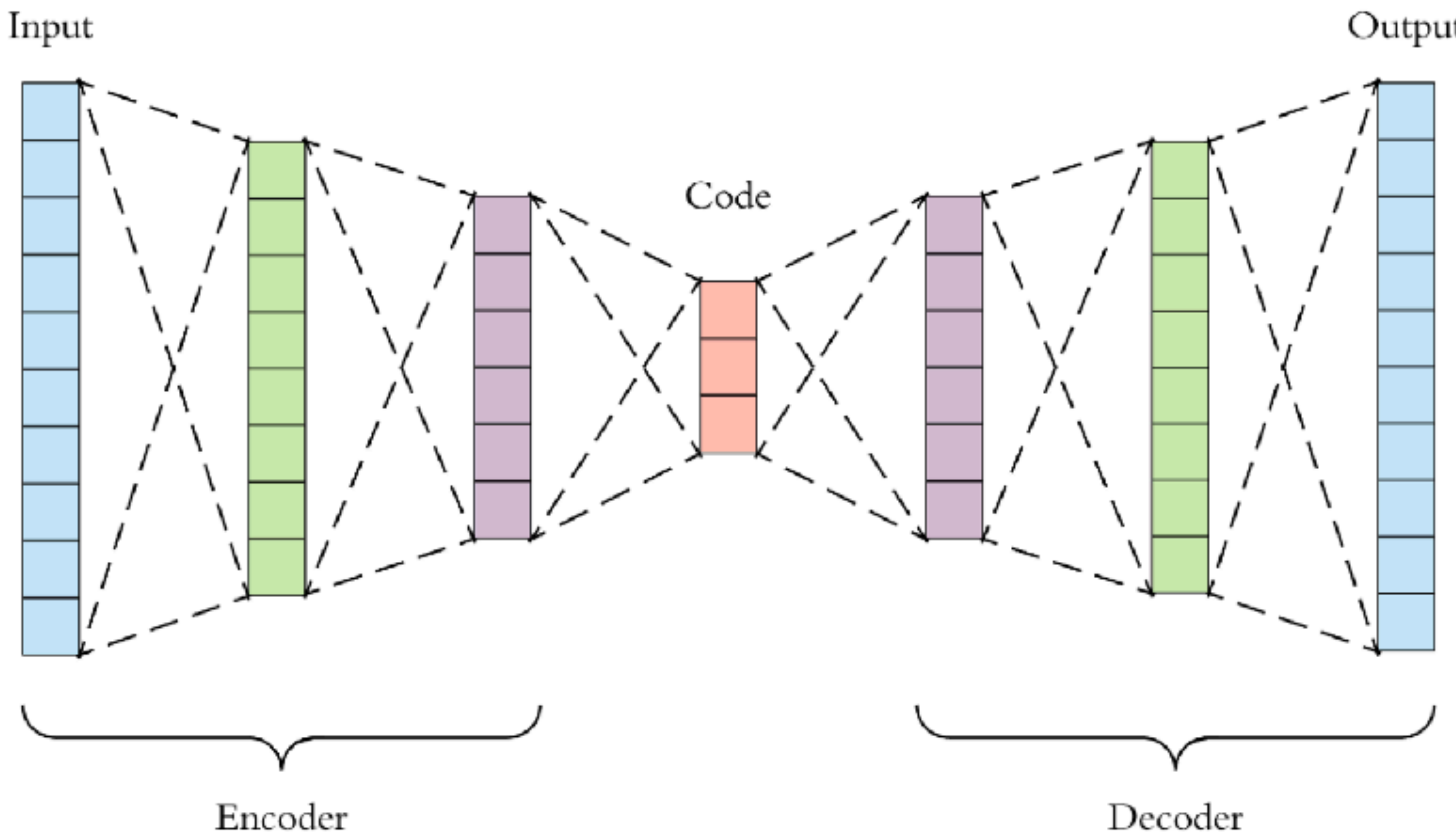
Auto-encoders



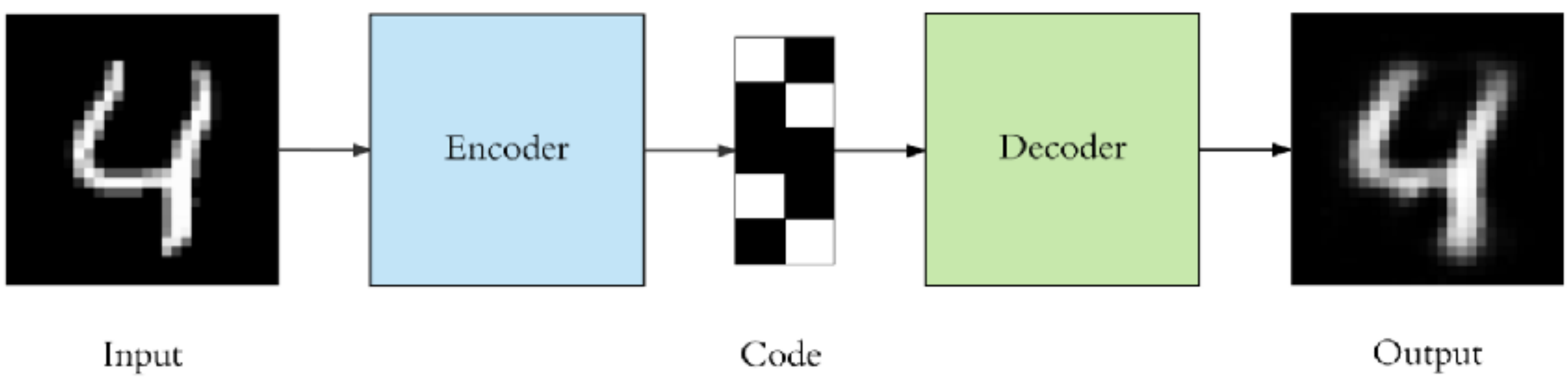
Output with the same shape as the input

Application ?

Dense auto-encoders



PCA/EOF

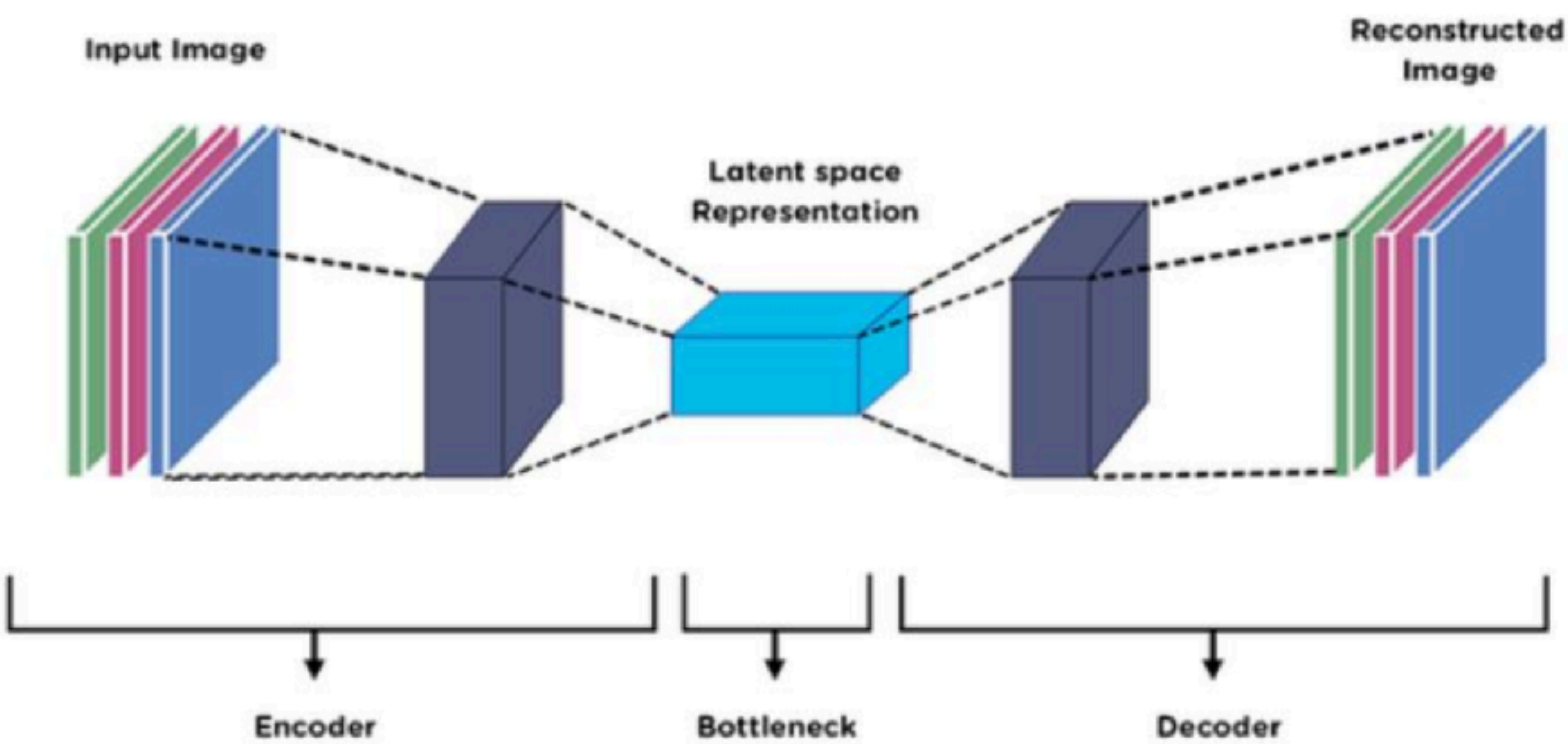


$$X \xrightarrow[Z = A \cdot X]{} Z \xrightarrow[\tilde{X} = A^t \cdot Z]{} \tilde{X}$$

PCA as a linear auto-encoder architecture.

Which additional constraint ?

Convolutional auto-encoders

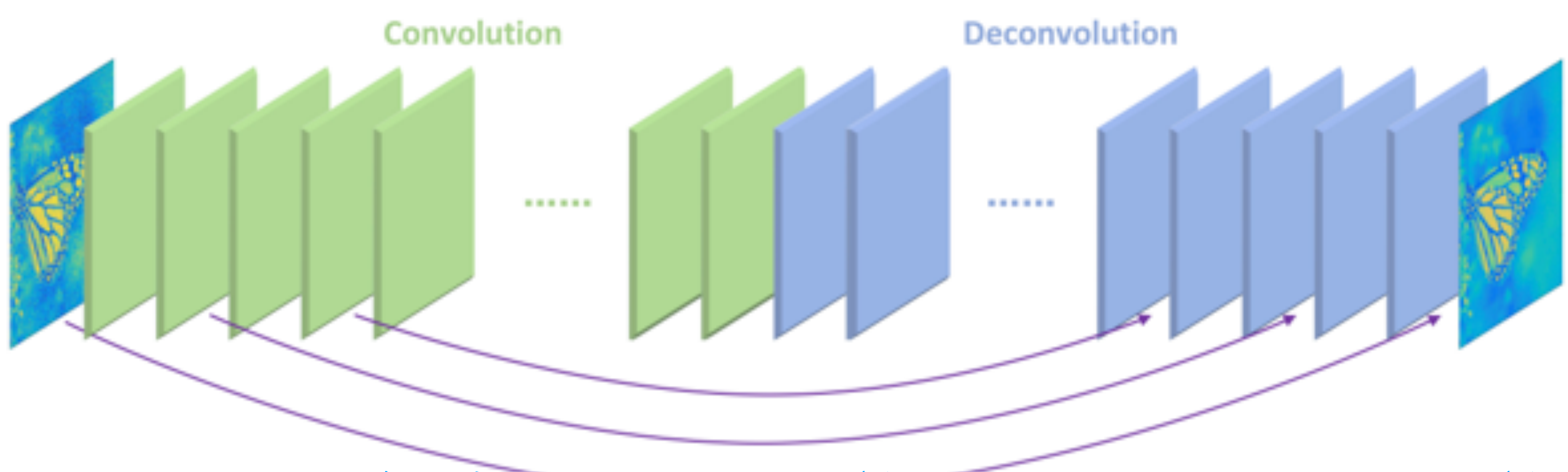


(Generator)

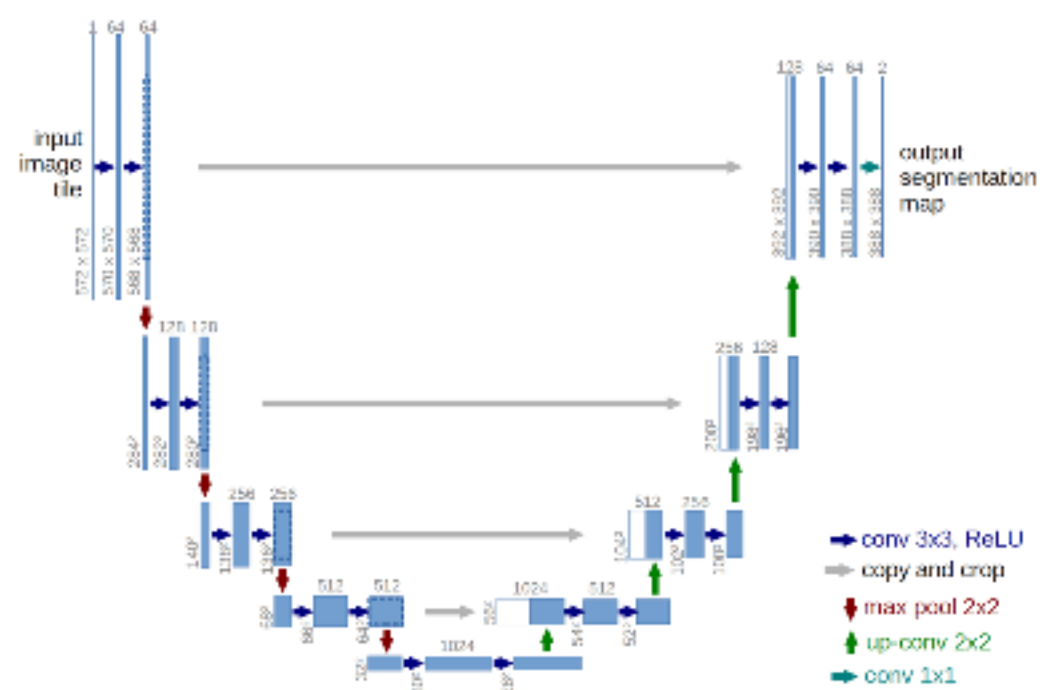
Source: https://www.analyticsvidhya.com/

Convolutional AE Zoo

Many applications do not require a low-dimensional representation
(e.g., densoising, interpolation, super-resolution,….)



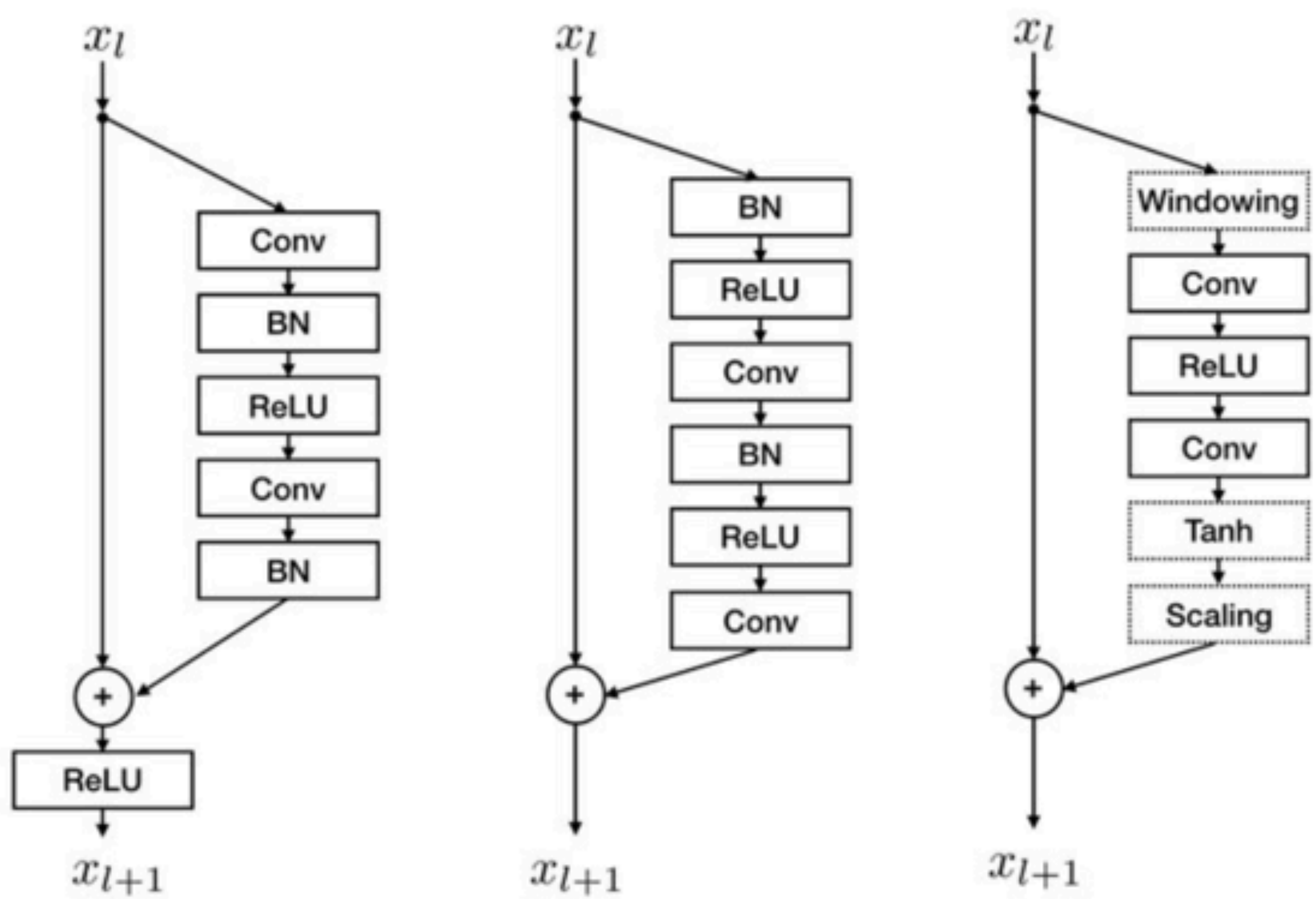
https://arxiv.org/pdf/1606.08921.pdf



UNet

Ronneberger et al., 2015

Convolutional AE Zoo



Residual Block (ResNet)

Rousseau et al., 2019

Often used to address vanishing gradients ("very" deep networks)

Lecture. #2
Things to know (AE)

- Auto-encoder
- Latent variable
- UNet
- ResNet